RDMA OVER CONVERGED ETHERNET INTEGRATION & TESTING on LS2085A ISSD

Rev A0-04

24th March, 2016

Implementation of Soft RDMA over Converged Ethernet on LS2085A ISSD, Link tests, Benchmark measurements and File Operations on SCSI disk using ISER protocol

Freescale Semiconductor Inc.

Revision History:

Date	Rev No.	Description	By
21-December-2015	A0_02	Soft-RoCE Implementation , link testing and benchmark measurements on LS2085A ISSD and file operations on SCSI disk using ISER protocol	VVDN Technologies
05-March-2016	A0_03	Added noop timeout change	VVDN Technologies
24-March-2016	A0_04	Added Soft RDMA over Ethernet (ROCE) driver for x86	VVDN Technologies

Table of Contents

Table of Figures

1 INTRODUCTION

This document describes the steps required for establishing and maintaining the RDMA – Remote Direct Memory Access connection between two endpoints by adding Soft-RoCE support to LS2085A ISSD. RDMA is achieved on LS2085A ISSD using software itself without utilizing any specialized hardware. Hence the specialized hardware for establishing RDMA is no more mandatory.

1.1 Scope

Establishing the RDMA link between LS2085A ISSD and other end point which may be x86 system or LS2085AISSD and ensuring the connectivity and performance over the link. Performing the block operations on the SCSI disks mounted using ISER protocol. Following are tasks involved

- Ensuring the RDMA between two endpoints using "rping" link test.
- Benchmarking the Soft-RoCE for analyzing its performance using benchmark tools.
- Mounting the SCSI disk/drive via USB 3.0 or RAM disk and to perform the file operations using ISER protocol.

2 Prerequisites of setup

2.1 Hardware setup

Following are the necessary hardware requirements

- LS2085A ISSD
- X86 system – in case of testing between LS2085AISSD and x86
- SFP modules and cables
- USB based storage disk (say Hard disk)

2.2 Software setup

Following are the necessary software packages

- OFED – 1.5.2 –rxe package (*included in the iso image*)
- Open –iscsi-2.0-873 (*included in the iso image*)
- Targetcli-2.0rc1 (*included in the iso image*)

3 OFED-1.5.2-rxe Packages

RDMA over converged ethernet can be achieved by using specialized RDMA NIC hardware. Instead it can be achieved by software itself. The distribution released by the Open fabrics is used to achieve the RDMA over converged ethernet using software termed as Soft-RoCE. Soft-RoCE uses the upper layers of the Infiniband and the datalink layer is swapped with the Ethernet data link layer and physical layer with dedicated ethernet physical layer. Hence RoCE is IBoE – Infiniband over Ethernet.
OFED (Open Fabrics Enterprise Distribution) -1.5.2 – rxe includes

- Kernel-level drivers
- Channel oriented RDMA and send/receive operations
- Kernel bypass
- both kernel and user level API
- Sockets data exchange (e.g. RDS, SDP)
- NAS & SAN (e.g. iSER, SRP, NFS-RDMA)

3.1 Libibverbs

It is a user space library which implements the verbs abstraction for using RDMA in software. Most RDMA programs are developed over this library, so it is quite mandatory to install it.

Package Name	Description
libibverbs	Libibverbs is a library that allows user space processes to use RDMA "verbs" as described in the InfiniBand Architecture Specification and the RDMA Protocol Verbs Specification. This includes direct hardware access from user space to InfiniBand/iWARP adapters (kernel bypass) for fast path operations.
libibverbs-utils	Contains libibverbs utils such as ibv_devinfo for displays information about RDMA devices.
libibverbs-devel	Header files for the libibverbs library.

3.2 DAPL

DAPL is a library which implements common API for RDMA protocols. It can be used for specific MPI implementation.

Package Name	Description
Dapl	libdat and libdapl provide a user space implementation of the DAT 2.0 API and is built to natively support InfiniBand/iWARP network technology.
dapl-utils	Useful test suites to validate the dapl library API's and operation.
compat-dapl	The DAT programming API provides a means of utilizing high performance network technologies, such as InfiniBand and iWARP, without needing to write program to use those technologies directly. This package contains the libraries that implement version 1.2 of the DAT API. The current (and recommended version for any new code) is 2.0. These 1.2 libraries are provided solely for backward compatibility.
dapl-devel	Header files for libdat and libdapl library.

3.3 Performance tools

Benchmark tools for RDMA (InfiniBand, RoCE and iWARP) subnets.

Package Name	Description
Perftest	Perftest is a collection of simple test programs designed to utilize RDMA communications and provide performance numbers over those RDMA connections. It does not work on normal TCP/IP networks, only on RDMA networks.
qperf	Measure socket and RDMA performance.

3.4 Hardware user space drivers, libraries and tools

These are the hardware-specific user space libraries (and tools) for various RDMA devices. Packages for appropriate hardware need to be installed.

Package Name	Description
libmlx5	libmlx5 provides a device-specific userspace driver for Mellanox Connect-IB HCAs for use with the libibverbs library.
libmlx4	libmlx4 provides a device-specific userspace driver for Mellanox ConnectX HCAs for use with the libibverbs library.
libmthca	libmthca provides a device-specific userspace driver for Mellanox HCAs (MT23108 InfiniHost and MT25208 InfiniHost III Ex) for use with the libibverbs library.
mstflint	This package contains a burning tool for Mellanox manufactured HCA cards. It also provides access to the relevant source code.
libnes	Userspace hardware driver for use with the libibverbs InfiniBand/iWARP verbs

	library. This driver enables NetEffect iWARP capable ethernet devices
libipathverbs	libipathverbs provides a device-specific userspace driver for QLogic HCAs for use with the libibverbs library.
infinipath-psm	The PSM Messaging API, or PSM API, is QLogic's low-level user-level communications interface for the Truescale family of products. PSM users are enabled with mechanisms necessary to implement higher level communications interfaces in parallel environments.

3.5 Subnet Management (Infiniband Only)

Package Name	Description
opensm	OpenSM is the OpenIB project's Subnet Manager for Infiniband networks. The subnet manager is run as a system daemon on one of the machines in the infiniband fabric to manage the fabric's routing state. This package also contains various tools for diagnosing and testing Infiniband networks that can be used from any machine and do not need to be run on a machine running the opensm daemon.
ibsim	ibsim provides simulation of infiniband fabric for using with OFA OpenSM, diagnostic and management tools.
libibmad	libibmad provides low layer IB functions for use by the IB diagnostic and management programs. These include MAD, SA, SMP, and other basic IB functions.
libibumad	libibumad provides the user MAD library functions which sit on top of the user MAD modules in the kernel. These are used by the IB diagnostic and management tools, including OpenSM.
libibmad-devel	Development files for the libibmad library.
libibumad-devel	Development files for the libibumad library.
opensm-libs	Shared libraries for Infiniband user space access.
libibcommon	libibcommon provides common utility functions for the OFA diagnostic and management tools.

3.6 Diagnostics

Diagnostics tools for InfiniBand subnet.

Package Name	Description

infiniband-diags	This package provides IB diagnostic programs and scripts needed to diagnose an IB subnet.
ibutils	ibutils provides IB network and path diagnostics.
ibutils-libs	Shared libraries used by the Mellanox Infiniband diagnostic utilities

3.7 Communication Management

Package Name	Description
librdmacm	librdmacm provides a userspace RDMA Communication Management API.
librdmacm-utils	Example test programs for the librdmacm library.
librdmacm-devel	Development files for the librdmacm library.
ibacm	The ib_acm daemon helps reduce the load of managing path record lookups on large InfiniBand fabrics by providing a user space implementation of what is functionally similar to an ARP cache. The use of ib_acm, when properly configured, can reduce the SA packet load of a large IB cluster from O(n^2) to O(n). The ib_acm daemon is started and normally runs in the background, user applications need not know about this daemon as long as their app uses librdmacm to handle connection bring up/tear down. The librdmacm library knows how to talk directly to the ib_acm daemon to retrieve data.

3.8 User-level tools for ULP (Upper Layer Protocol)

Package Name	Description
srptools	In conjunction with the kernel ib_srp driver, srptools allows you to discover and use SCSI devices via the SCSI RDMA Protocol over InfiniBand.
rds-tools	Various tools for support of the RDS (Reliable Datagram Socket) API. RDS is specific to InfiniBand and iWARP networks and does not work on non-RDMA.

Note: Currently “libibverbs” , “librdmacm” and “librxe” are only included. Based on the requirements others can be enabled in the ***“<yocto_directory>/LS2085********/ meta-fsl-networking/recipes-extended/ofed-1.5.2/ofed-1.5.2/ofed_beta_config.conf”***

4 Building Instructions for LS2085AISSD on Yocto

- Mount the *.iso* image given which is usually in the format of "*Layerscape2-SDK-YYYYMMDD-yocto.iso*".

```
# mount –o loop <iso_image> <mount_dir>
```

```
For example
# mount –o loop Layerscape2-SDK-20150625-yocto.iso  /home/user/mount_dir
```

- Once the image is mounted, go to the mounted directory and run "*install*".

```
# ./install
```

Note :

During installation user will be prompted to give "*Yocto build directory location* " (Default location is */home/<user>/ Layerscape2-SDK-YYYYMMDD-Yocto*).

- Once Yocto build directory is created, go to the build directory and execute the below mentioned command to setup the *build environment for Yocto.*

```
#sh poky/scripts/host_prepare.sh
# source poky/fsl-setup-poky -m ls2085aissd  [-j jobs] [-t tasks] [-l] [-h]
```

Option	Description
-m	Specify machine name
-j	Number of jobs for makes to spawn during the compilation stage.
-t	Number of parallel BitBake tasks that can be issued.
-d	non-default path of DL_DIR (downloaded source)
-c	non-default path of SSTATE_DIR (shared state Cache)
-b	non-default path of project folder (build_${machine}_release)
-l	lite mode. To help conserve disk space, deletes the building directory once the package is built.
-s	appends an extra string to project folder.
-p	appends fsl cache and source mirrors (For FSL Internal Use Only)
-h	Help

For example,

#source poky/fsl-setup-poky -m ls2085aissd –j8 –t8

- To build *.itb* image , execute the following commands

```
# bitbake <-v> fsl-image-core
```

```
# bitbake <–v> fsl-image-kernelitb
```

- ***kernel-ls2085aISSD-yyyymmddHHMMSS.itb*** file will be created in the below location.

For Example

```
# cd <yocto_directory>/Layerscape2-SDK-20150515-
yocto/build_ls2085aissd_release/tmp/deploy/images/
```

Note:
* If ***.itb*** image is build on 17/06/2015 @ 13:02:52 then,
.itb file is created *as* ***kernel-ls2085aissd-20150617130252.itb***

5 RoCE link test setup

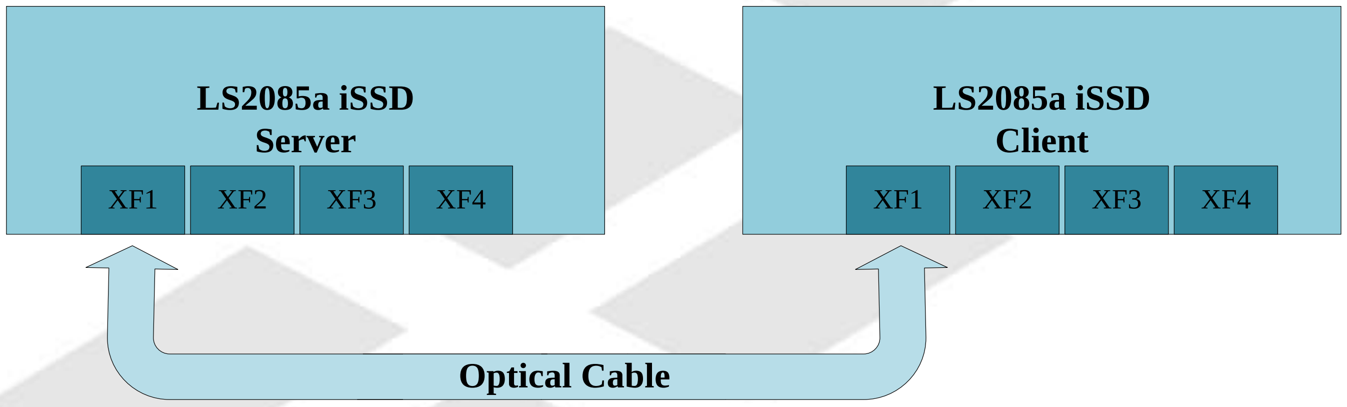


Figure 1 – Test setup for RoCE link test

- Boot the LS2085a iSSD target board with the *".itb"* file generated from the build.

- Mount */sys/kernel/config* as type *configfs.*

```
# mount –t configfs none /sys/kernel/config
```

- Make the interface up.

```
# ifconfig <interface-name>  <ip-address> up
```

- Once the board is up, "*modprobe*" the following infiniband core modules.

```
#modprobe ib_addr

# modprobe ib_core

# modprobe ib_mad

#modprobe ib_sa

#modprobe ib_cm

#modprobe iw_cm

# modprobe rdma_cm

# modprobe ib_umad

# modprobe ib_uverbs

# modprobe ib_ucm

# modprobe rdma_ucm

#modprobe ib_iser

#modprobe ib_isert
```

- Then go to the rxe directory and "*modprobe*" rxe modules.

```
# cd ../hw/rxe

# modprobe ib_rxe.ko
```

- Add the network interface as parameter for RoCE module using following command.

```
# echo <interface> > /sys/module/ib_rxe/parameters/add
```

```
For Example
For 10G link SFP module
# echo ni1 > /sys/module/ib_rxe/parameters/add

For 1 G link using Ethernet 1G Network module
# echo eth0 > /sys/module/ib_rxe/parameters/add
```

- Give the following command to enable the RoCE version

```
# mkdir –p /sys/kernel/config/rdma_cm/rxe0

# echo IBOE V2 | tee /sys/kernel/config/rdma_cm/rxe0/default_roce_mode

# rmdir /sys/kernel/config/rdma_cm/rxe0
```

- To get the list of RDMA devices

```
# ibv_devices
```

- To query RDMA devices

```
# ibv_devinfo
```

6 Instructions to run link functionality test (rping)

"*rping*" is the RDMA CM connection and RDMA ping-pong test. It establishes a reliable RDMA connection between two nodes using the *librdmacm*, optionally performs RDMA transfers between the nodes, and then disconnects.

6.1 rping "ping/pong" loop

- Client sends source rkey/addr/len
- Server receives source rkey/addr/len
- Server rdma reads "ping" data from the source
- Server sends "go ahead" on rdma read completion
- Client sends sink rkey/addr/len
- Server receives sink rkey/addr/len
- Server rdma writes "pong" data to the sink
- Server sends "go ahead" on rdma write completion
- <repeat loop>

6.2 Syntax for rping

6.2.1 Server side

```
# rping –s [-v] [-V] [-d] [-P] [-a address] [-p port] [-C message count] [-S message size]
```

6.2.2 Client side

```
# rping –c [-v] [-V] [-d] -a address [-p port] [-C message count] [-S message size]
```

6.2.3 Options

Option	Description
-s	Run as server
-c	Run as client
-a	On the server, specifies the network address to bind the connection to On the client, specifies the server address to connect to
-p	Port number for listening server
-v	Display ping data
-V	Validate ping data
-d	Display debug information
-C	Message count – the number of messages to transfer over the connection (default infinite)
-S	Message size – Size of each message to transfer over each connection. (default infinite)
-P	Run the server in the persistent mode. This allows multiple rping clients to connect to a single server instance. The server will run until killed.

6.3 rping example and output

6.3.1 Server side

```
# rping –s –a ::0 -vV
```

6.3.2 Console output

```
server ping data: rdma-ping-0: ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqr
server ping data: rdma-ping-1: BCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrs
server ping data: rdma-ping-2: CDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrst
server ping data: rdma-ping-3: DEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstu
server ping data: rdma-ping-4: EFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuv
server ping data: rdma-ping-5: FGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvw
server ping data: rdma-ping-6: GHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwx
server ping data: rdma-ping-7: HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxy
server ping data: rdma-ping-8: IJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
server ping data: rdma-ping-9: JKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyzA
```

6.3.3 Client side

```
# rping –c –C 10 –a fe80::3617:fec7:f66b -vV
```

6.3.4 Console output

```
ping data: rdma-ping-0: ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqr
ping data: rdma-ping-1: BCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrs
ping data: rdma-ping-2: CDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrst
ping data: rdma-ping-3: DEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstu
ping data: rdma-ping-4: EFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuv
ping data: rdma-ping-5: FGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvw
ping data: rdma-ping-6: GHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwx
ping data: rdma-ping-7: HIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxy
ping data: rdma-ping-8: IJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
ping data: rdma-ping-9: JKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyzA
```

7 Instructions to run performance test (Bandwidth & Latency measurement)

"*Perftest (Performance test)*" package is the collection of tests written over uverbs intended for use as a *performance micro-benchmark*. As an example, the tests can be used for Hardware or Software tuning and/or functional testing.
The collection contains the set of Bandwidth and Latency benchmark such as

- **Read** - ib_read_bw and ib_read_lat.
- **Write** - ib_write_bw and ib_write_lat.
- **Send** - ib_send_bw and ib_send_lat.
- **RDMA** - rdma_bw and rdma_lat.
- **In additional** : ib_write_bw_postlist and ib_clock_test.

7.1 Syntax for perf test

7.1.1 Server side

```
# ./<test name> <options>
```

7.1.2 Client side

```
# ./<test name> <options> <server IP address>
```

7.1.3 Options

Options	Description

-p, --port=<port>	Listen on/connect to port <port> (default: 18515).
-m, --mtu=<mtu>	Mtu size (default: 1024).
-d, --ib-dev=<dev>	Use IB device <dev> (default: first device found)
-i, --ib-port=<port>	Use port <port> of IB device (default: 1).
-s, --size=<size>	Size of message to exchange (default: 1).
-a, --all	Run sizes from 2 till 2^23
-t, --tx-depth=<dep>	Size of tx queue (default: 50).
-r, --rx-depth=<dep>	Make rx queue bigger than tx (default 600).
-n, --iters=<iters>	Number of exchanges (at least 100, default: 1000)
-I, --inline_size =<size>	Max size of message to be sent in inline mode On Bw tests default is 1, latency tests is 400.
-C, --report-cycles	Report times in cpu cycle units.
-u, --qp-timeout=<timeout>	QP timeout, timeout value is 4 usec*2 ^ (timeout). Default is 14.
-S, --sl=<sl>	SL (default 0)
-H, --report-histogram	Print out all results (Default: summary only). Only on Latency tests
-x, --gid-index=<index>	Test uses GID with GID index taken from command Line (for RDMAoE index should be 0).
-b, --bidirectional	Measure bidirectional bandwidth (default unidirectional) On BW tests only (Implicit on latency tests)
-V, --version	Display version number.
-e, --events	Sleep on CQ events (default poll).
-N, --no peak-bw	Cancel peak-bw calculation (default with peak-bw)
-F, --CPU-freq	Do not fail even if cpufreq_ondemand module.

7.2 Sample Bandwidth measurement using “ib_read_bw”

7.2.1 Server side

```
# ./ib_read_bw
```

7.2.2 Console output

```
                    RDMA_Read BW Test
 Connection type : RC
 Link type is ETH
 Number of outstanding reads is 4
 local address: LID 0000 QPN 0x0011 PSN 0xf5ebc2 OUT 0x04 RKey 0x000401 VAddr
0x007fb5129d6000
 GID: 254:128:00:00:00:00:00:00:242:77:162:255:254:104:177:223
remote address: LID 0000 QPN 0x0011 PSN 0x477b6 OUT 0x04 RKey 0x000480 VAddr
0x000000020f7000
 GID: 254:128:00:00:00:00:00:00:54:23:235:255:254:112:102:187
 Mtu : 1024
------------------------------------------------------------------
```

7.2.3 Client side

```
# ./ib_read_bw -s 4096 -n 15000 192.168.1.1
```

7.2.4 Console output

```
---------------------------------------------------------------
                    RDMA_Read BW Test
 Connection type : RC
 Link type is ETH
 Using gid index 0 as source GID
 Number of outstanding reads is 4
 local address: LID 0000 QPN 0x0011 PSN 0x8caac5 OUT 0x04 RKey 0x000408 VAddr
0x00000001f55000
 GID: 254:128:00:00:00:00:00:00:242:77:162:255:254:104:177:223
remote address: LID 0000 QPN 0x0011 PSN 0xed67fc OUT 0x04 RKey 0x000445 VAddr
0x007f97401d6000
 GID: 254:128:00:00:00:00:00:00:54:23:235:255:254:112:102:187
 Mtu : 1024
---------------------------------------------------------------
 #bytes     #iterations    BW peak[MB/sec]    BW average[MB/sec]
 4096       15000          105.93             103.69
---------------------------------------------------------------
```

7.4 Sample latency measurement using “ib_read_lat”

7.4.1 Server side

```
# ./ib_read_lat
```

7.4.2 Console output

```
---------------------------------------------------------------
                    RDMA_Read Latency Test
 Connection type : RC
 Link type is ETH
 Using gid index 0 as source GID
 Number of outstanding reads is 4
 local address: LID 0000 QPN 0x0011 PSN 0x8e759b OUT 0x04 RKey 0x000422 VAddr
0x00000001dbb000
 GID: 254:128:00:00:00:00:00:00:242:77:162:255:254:104:177:223
remote address: LID 0000 QPN 0x0011 PSN 0xccc15f OUT 0x04 RKey 0x000416 VAddr
0x00000002044000
 GID: 254:128:00:00:00:00:00:00:54:23:235:255:254:112:102:187
Mtu : 1024
---------------------------------------------------------------
```

7.4.3 Client side

```
# ./ib_read_lat  -s 4096 -n 15000 192.168.1.1
```

7.4.4 Console output

```
---------------------------------------------------------------------
                RDMA_Read Latency Test
 Connection type : RC
 Link type is ETH
 Using gid index 0 as source GID
 Number of outstanding reads is 4
 local address: LID 0000 QPN 0x0011 PSN 0x8e759b OUT 0x04 RKey 0x000422 VAddr
0x00000001dbb000
 GID: 254:128:00:00:00:00:00:00:242:77:162:255:254:104:177:223
remote address: LID 0000 QPN 0x0011 PSN 0xccc15f OUT 0x04 RKey 0x000416 VAddr
0x00000002044000
 GID: 254:128:00:00:00:00:00:00:54:23:235:255:254:112:102:187
---------------------------------------------------------------------
 #bytes #iterations    t_min[usec]    t_max[usec]  t_typical[usec]
 4096    15000         91.44         638.42       102.53
---------------------------------------------------------------------
```

8 File operations on SCSI disk using ISER protocol

8.1 Introduction of ISER

ISER – iSCSI Extension for RDMA is a computer network protocol that extends the Internet Small Computer System Interface (iSCSI) to use the Remote Direct Memory Access. Generally iSCSI uses TCP/IP stack. But in the case of RDMA the TCP stack should not be included. Instead of TCP stack, RoCE uses Infiniband stack and hence it provides the low latency.
In RoCE, ISER is one of the Upper Layer Protocol and it permits data to be transferred directly into and out of SCSI memory buffers without intermediate copies.
ISER can be simply explained as the emulation layer that translates iSCSI to RDMA transactions. ISER transports iSCSI control messages in RDMA SEND/RECV commands and transports iSCSI payload in RDMA READ/WRITE operations.

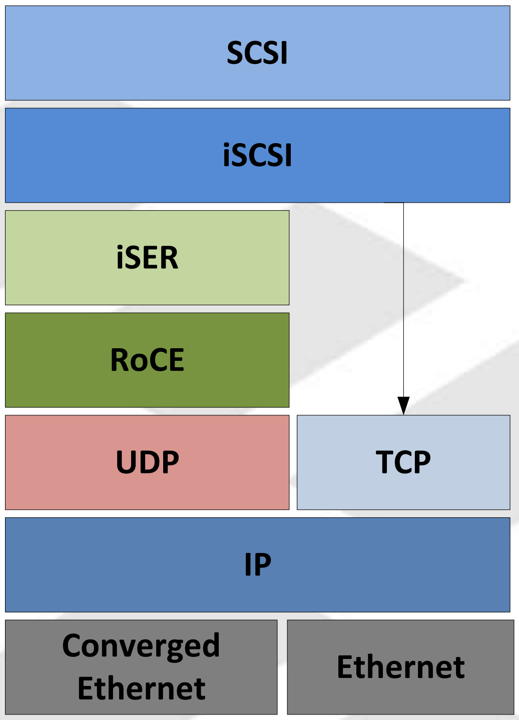


Figure 2 – iSER – ISCSI Layer Comparision

8.2 Test setup for ISER

Following is the test setup for the ISER

- Validate the availability SCSI disk in the target side.
- Ensure the connectivity between two boards.

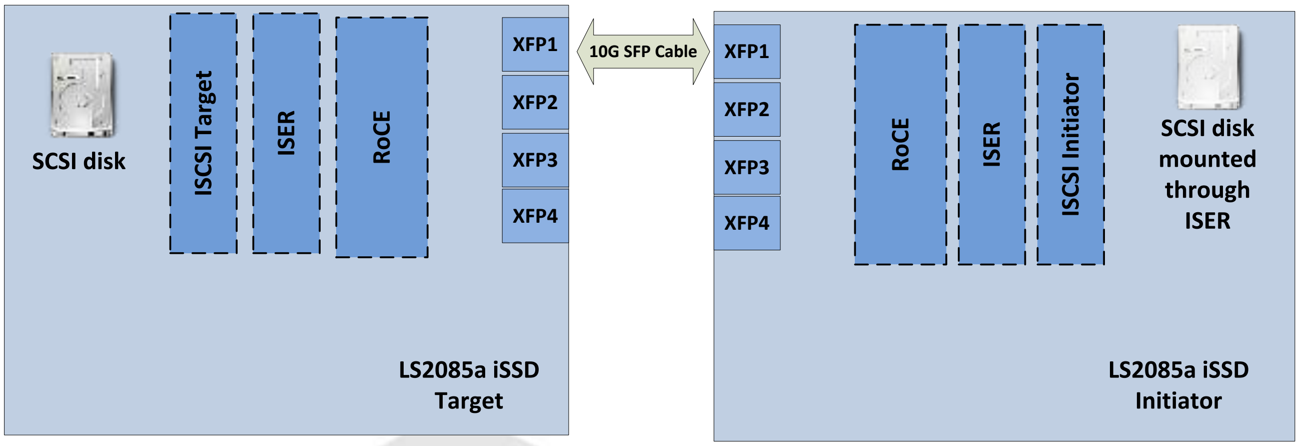


Figure 3 – Test setup for ISER

*Note – Make sure the target XFPx and initiator XFPx have different MAC addresses.

8.3 Procedure

8.3.1 Target side

- Make sure “ib_isert.ko” and “ib_iser.ko” are inserted.
- Go to the directory “/etc/targetcli-2.0rc1”

```
# cd /etc/targetcli-2.0rc1
```

- Run the command *“# PYTHONPATH=. ./scripts/targetcli”*

```
# PYTHONPATH=. ./scripts/targetcli
```

- In the targetcli user space, give the below mentioned command for reserving the disk to get mounted on the initiator when it gets login successfully

```
>/backstores/iblock create name=<device_name> dev=<block-device>

Example:

>/backstores/iblock create name=iser_disk dev=/dev/sda1
```

- In case of using the NVMe SSD, then add the "TYPE_DISK" value, simply the major number to the *"/usr/lib/python2.7/site-packages/rtslib/utils.py"* in the "*type_disk_known_majors*" structure.

  For example: update the structure with **259 # NVMe SSD** at the end of the structure.

- In case of using RAM as the SCSI disk to be mounted, then use "rd_mcp" in /backstores

>/backstores/rd_mcp create name=<device_name> size=<size>

Example:

>/backstores/rd_mcp create name=iser_disk size=1GB

- Targetcli automatically creates "Target Portal Group – TPG" as per default assigns sequentially increasing TPG tags, starting from "1" thereby creating a TPG1.

>/iscsi create

Output :

```
/backstores/iblock/my_disk> /iscsi create
Created target iqn.2003-01.org.linux-iscsi.san01.x8664:sn.05135a0e4a11.
Selected TPG Tag 1.
Successfully created TPG 1.
Entering new node /iscsi/iqn.2003-01.org.linux-iscsi.san01.x8664:sn.05135a0e4a11/tpgt1.
/iscsi/iqn.20...a0e4a11/tpgt1>
```

- Now add the LUN (Logical Unit Number) to the iSCSi target

>cd /iscsi/ iqn.20…a11/tpgt1

/iscsi/iqn.20…a11/tpgt1 > luns create /backstores/iblock/iser_disk

Output:

```
/iscsi/iqn.20...a0e4a11/tpgt1> luns/ create /backstores/iblock/iser_disk
Selected LUN 0.
Successfully created LUN 0.
Entering new node /iscsi/iqn.2003-01.org.linux-
iscsi.san01.x8664:sn.05135a0e4a11/tpgt1/luns/lun0.
/iscsi/iqn.20...gt1/luns/lun0>
```

- Now create the network portal. Assign an active IP address to ISCSI TPG to form a valid iSCSI endpoint.

/iscsi/iqn.20…a11/tpgt1 > portals create <ip-address>

Output

```
/iscsi/iqn.20...a0e4a11/tpgt1> portals/ create 192.168.1.20
Using default IP port 3260
Successfully created network portal 192.168.1.20:3260.
Entering new node /iscsi/iqn.2003-01.org.linux-
iscsi.san01.x8664:sn.05135a0e4a11/tpgt1/portals/192.168.1.20:3260.
/iscsi/iqn.20...68.1.139:3260>
/iscsi/iqn.20...68.1.139:3260> cd <
Taking you back to /iscsi/iqn.2003-01.org.linux-iscsi.san01.x8664:sn.05135a0e4a11/tpgt1.
/iscsi/iqn.20...a0e4a11/tpgt1>
```

*** Incase if the IP address is not mentioned then the active IP address is chosen automatically.

- For enabling the ISER operation give "iser_enable" in the respective portals created.

  /iscsi/iqn.20…a11/tpgt1 > portals /<ip-address>/ iser_enable

Output:

```
/iscsi/iqn.20...68.1.139:3260> iser_enable
iser operation has been enabled
/iscsi/iqn.20....100.0.1:3260> ls
o- 192.168.1.139:3260 .......................................... [OK, iser enabled]
/iscsi/iqn.20...68.1.139:3260> cd <
Taking you back to /iscsi/iqn.2003-01.org.linux-iscsi.san01.x8664:sn.05135a0e4a11/tpgt1.
/iscsi/iqn.20...a0e4a11/tpgt1>
```

- Configure the access rights to allow logins from the initiators. Here described DEMO mode only for ISER demo purpose.

- For demo mode, disable all authentication

  /iscsi/iqn.20…a11/tpgt1 > set attribute authentication=0 demo_mode_write_protect=0 generate_node_acls=1 cache_dynamic_acls=1

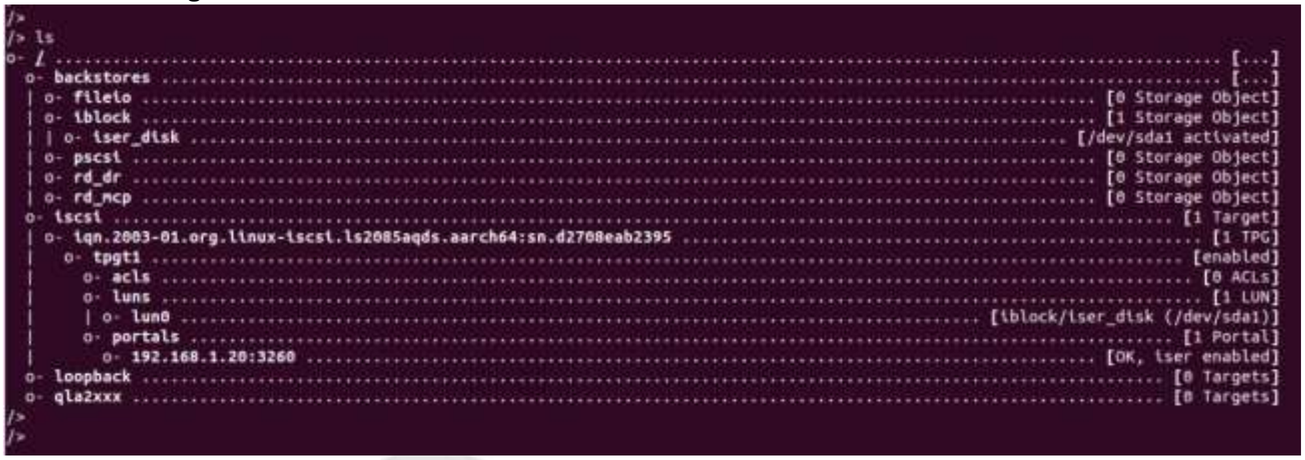
Output:

```
/iscsi/iqn.20...a0e4a11/tpgt1> set attribute authentication=0 demo_mode_write_protect=0
generate_node_acls=1 cache_dynamic_acls=1.
Parameter demo_mode_write_protect is now '0'.
Parameter authentication is now '0'.
Parameter generate_node_acls is now '1'.
Parameter cache_dynamic_acls is now '1'.
/iscsi/iqn.20...a0e4a11/tpgt1> cd /
/>
```

- After all settings give “saveconfig” for saving the configurations.

```
/saveconfig
```

- The final configuration will be like



```
/>
/> ls
o- / ...................................................................................... [...]
  o- backstores ........................................................................... [...]
  | o- fileio ................................................................ [0 Storage Object]
  | o- iblock ................................................................ [1 Storage Object]
  | | o- iser_disk ......................................................... [/dev/sda1 activated]
  | o- pscsi ................................................................. [0 Storage Object]
  | o- rd_dr ................................................................. [0 Storage Object]
  | o- rd_mcp ................................................................ [0 Storage Object]
  o- iscsi ..................................................................... [1 Target]
  | o- iqn.2003-01.org.linux-iscsi.ls2085aqds.aarch64:sn.d2708eab2395 ..................... [1 TPG]
  |   o- tpgt1 ..................................................................... [enabled]
  |     o- acls ...................................................................... [0 ACLs]
  |     o- luns ....................................................................... [1 LUN]
  |     | o- lun0 ................................................ [iblock/iser_disk (/dev/sda1)]
  |     o- portals ................................................................. [1 Portal]
  |       o- 192.168.1.20:3260 .................................................. [OK, iser enabled]
  o- loopback ................................................................... [0 Targets]
  o- qla2xxx .................................................................... [0 Targets]
/>
/>
```

8.3.3 Initiator side

Following are the steps to be done in the Initiator side

- Make sure “ib_iser.ko” module is inserted.
- Give the following command for creating the *initiator name.*

```
#iscsi-name
iqn.2003-01.org.linux-iscsi.ls2085aISSD.aarch64:sn.2d3e7781568
```

- Update *“/etc/iscsi/initiatorname.iscsi”* with the name get from the *“iscsi-name”* command.

```
#echo "InitiatorName=iqn.2003-01.org.linux-iscsi.ls2085aISSD.aarch64:sn.2d3e7781568" >
/etc/iscsi/initiatorname.iscsi
```

- Change the noop timeout and startup settings as *“Automatic”* in *“/etc/iscsi/iscsid.conf”.*

```
#*****************
# Startup settings
#*****************

# To request that the iscsi initd scripts startup a session set to "automatic".
node.startup = automatic
#
# To manually startup the session set to "manual". The default is manual.
#node.startup = manual

node.conn[0].timeo.noop_out_interval = 120
node.conn[0].timeo.noop_out_timeout = 240
```

- Kill all *iscsid* and restart *iscsid*

```
#killall iscsid

#iscsid
```

- Give the following command for *"discovering"* the target

```
#iscsiadm –m discovery –t st –p <ip-address>
```

Example

```
#iscsiadm –m discovery –t st –p 192.168.1.20
192.168.1.20:3260,1 iqn.2003-01.org.linux-
iscsi.ls2085aISSD.aarch64:sn.d2708eb2395
```

- After discovering the target update the transport as "iser" by giving the following command.

```
#iscsiadm –m node –T <target_name> -p <ip-addr> -o update –n iface.transport_name –v
iser
```

Example

```
#iscsiadm –m node –T iqn.2003-01.org.linux-iscsi.ls2085aISSD.aarch64:sn.d2708eb2395
 –p 192.168.1.20 –o update –n iface.transport_name –v iser
```

- After updating the transport then login to the target

```
#iscsiadm –m node –T <target_name> -p <ip-addr> --login
```

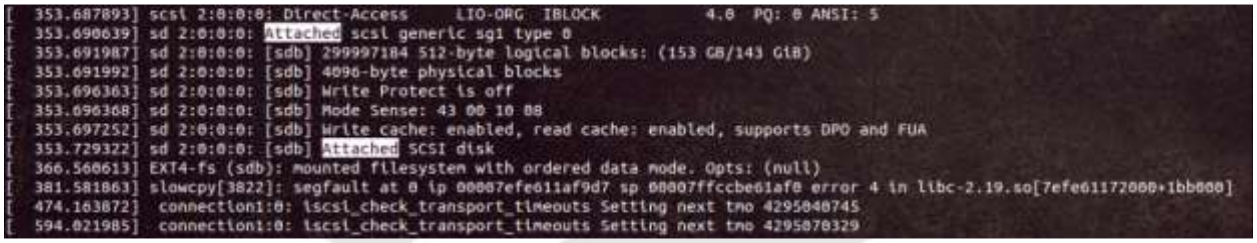
Example

#iscsiadm –m node –T iqn.2003-01.org.linux-iscsi.ls2085aISSD.aarch64:sn.d2708eb2395

–p 192.168.1.20 –-login

```
Logging in to [iface: default, target: iqn.2003-01.org.linux-
iscsi.ls2085aISSD.aarch64:sn.d2708eb2395, portal:
192.168.1.20,3260] (multiple)
Login to [iface: default, target: iqn.2003-01.org.linux-
iscsi.ls2085aISSD.aarch64:sn.d2708eb2395, portal:
192.168.1.20,3260] successful.
```

- The device is mounted in the initiator side. Check the dmesg for the appropriate device



```
[  353.687893] scsi 2:0:0:0: Direct-Access     LIO-ORG  IBLOCK           4.0  PQ: 0 ANSI: 5
[  353.690639] sd 2:0:0:0: Attached scsi generic sg1 type 0
[  353.691987] sd 2:0:0:0: [sdb] 299997184 512-byte logical blocks: (153 GB/143 GiB)
[  353.691992] sd 2:0:0:0: [sdb] 4096-byte physical blocks
[  353.696363] sd 2:0:0:0: [sdb] Write Protect is off
[  353.696368] sd 2:0:0:0: [sdb] Mode Sense: 43 00 10 08
[  353.697252] sd 2:0:0:0: [sdb] Write cache: enabled, read cache: enabled, supports DPO and FUA
[  353.729322] sd 2:0:0:0: [sdb] Attached SCSI disk
[  366.560613] EXT4-fs (sdb): mounted filesystem with ordered data mode. Opts: (null)
[  381.581863] slowcpy[3822]: segfault at 0 ip 00007efe611af9d7 sp 00007ffccbe61af0 error 4 in libc-2.19.so[7efe61172000+1bb000]
[  474.163872]  connection1:0: iscsi_check_transport_timeouts Setting next tmo 4295040745
[  594.021985]  connection1:0: iscsi_check_transport_timeouts Setting next tmo 4295070329
```

- Now mount the device

#mount /dev/sdb iser

- In case of using “rd_mcp” in the target side. Initiator should create the partition for mounted device, using the fdisk utils and make required file system using mkfs commands

# fdisk /dev/sdb

Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel

Building a new DOS disklabel with disk identifier 0x1ad15f73.

Changes will remain in memory only, until you decide to write them.

After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

```
Command (m for help): o
Building a new DOS disklabel with disk identifier 0x35247caf.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.
Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)
Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.
Syncing disks.
# fdisk /dev/sdb

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1):
Using default value 1
First sector (1-2097151, default 1):
Using default value 1
Last sector, +sectors or +size{K,M,G} (1-2097151, default 2097151):
Using default value 2097151
Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.
Syncing disks.
# ls /dev/sdb
sdb   sdb1
```

```
# mkfs.vfat /dev/sdb1
mkfs.fat 3.0.26 (2014-03-07)
#
```

- After mounting, the file transfer operations can be done.
- After block operations, unmount the device.

```
#umount iser
```

- Then logout from the target using the following command

```
#iscsiadm –m node –T <target_name> -p <ip-addr> --logout
```

Example

```
#iscsiadm –m node –T iqn.2003-01.org.linux-iscsi.ls2085aISSD.aarch64:sn.d2708eb2395
 –p 192.168.1.20 —-logout
```

8.3.4 Prevention of enqueue error in transmission

Three modes of transmission control is added as *"ib_rxe"* parameters, they are as follows (Only one mode is used at a time)

- None (Default)
- Watermark (Transmitted Packets – Enqueue Packets)
- Bandwidth (bits per second)

Step1: Control method needed is selected using *sysfs* entry "**control_method**"
Step2: Control value is assigned using *sysfs* entry "**control_value**"

```
#echo "<interface_name> <control_method>" > /sys/module/ib_rxe/parameters/control_method
#echo "<Interface_name><control_value>"   > /sys/module/ib_rxe/parameters/control_value
```

Examples:

```
#echo "ni1 none" > /sys/module/ib_rxe/parameters/control_method

#echo "ni1 watermark" > /sys/module/ib_rxe/parameters/control_method
#echo "ni1 45000" > /sys/module/ib_rxe/parameters/control_value

#echo "ni1 bandwidth" > /sys/module/ib_rxe/parameters/control_method
#echo "ni1 1000000000" > /sys/module/ib_rxe/parameters/control_value
```

9 Soft RDMA over Ethernet (RoCE) Driver for x86

9.1 Source

Kernel Space Driver is archived in the name of rxe-dev-master-next.tar.gz
User Space Library is archived in the name of librxe.tar.gz

9.2 Build Instructions

9.2.1 Compile and install kernel

- Extract the kernel tarball
  tar –xvzf rxe-dev-master-next.tar.gz
- To compile kernel, do the followings steps in order.
    1. cd rxe-dev-master-next
    2. cp /boot/config-$(uname –r) .config
    3. make menuconfig

Enable “Software RDMA over Ethernet (RoCE) driver” in category "Device Drivers -> Infiniband"
Enable CONFIG_INFINIBAND_ADDR_TRANS=y and CONFIG_INFINIBAND_ADDR_TRANS_CONFIGFS=y in new config file .config

4. make –j 32
5. make modules_install
6. make install
7. make headers_install INSTALL_HDR_PATH=/usr

- Verify that the new kernel entry is added (e.g. to grub); if not, need to add it manually.
- Boot with new kernel.

9.2.2 Install user space library

1. Install the following package: (example for ubuntu)
    1. Sudo apt-get install libswitch-perl (might vary according to distribution)
2. Make sure that the following upstream user space libraries are installed:
    1. libibverbs
    2. libibverbs-devel
    3. libibverbs-utils
    4. librdmacm
    5. librdmacm-devel
    6. librdmacm-utils

3. Compile and install user space library librxe by doing the following steps in order:
    - tar –xvzf librxe.tar.gz
    - cd librxe-dev

- ./configure --libdir=/usr/lib64/ --prefix=
- make
- make install

9.2.3 Configure Soft-RoCE (RXE):

1. Load ib_rxe kernel module using the rxe_cfg script included in the librxe RPM:
   rxe_cfg start (this might require sudo or root privileges)
2. Create RXE device over network interface (e.g. eth0):
   rxe_cfg add eth0
3. Use the status command to display the current configuration:
   rxe_cfg status
4. If configured successfully, user should see output similar to the following:

```
Name   Link   Driver     Speed    NMTU  IPv4_addr  RDEV    RMTU
 eth0   yes   mlx4_en                               rxe0    1024  (3)
```

Now user has an Infiniband device called “rxe0” that can be used to run any RoCE app.