

Mellanox Switch Conversion Guide v1.1

Written by mpogr on the STH forum

Updated by Necrotyr on the STH forum

About this guide

I'm writing this guide to help people with converting their managed SwitchX/X2 devices from some form of a "dumb" firmware (e.g. from EMC) into a real thing running MLNX-OS.

At the time of writing the first revision of this guide (Aug 5th 2016), this is still work in progress, as some of the fine tunings have not been done yet. Still, the device is 99% functional.

Also, many of the processes described below are manual and can be automated to a significant degree. I'll either do it in the future or, hopefully, someone else will share the burden ☺

This process is for techy-savvy people! If you're not comfortable with Linux command prompt, you probably shouldn't be messing around with this.

IF YOU DO SOMETHING WRONG, THERE IS A CHANCE YOU'LL END UP WITH A BRICKED DEVICE. YOU HAVE BEEN WARNED!

Software, licensing and legal/moral aspects

MLNX-OS is defined by Mellanox as a licensed product that is intended to be used on designated devices sold by Mellanox and its OEM partners.

BY FOLLOWING THIS PROCEDURE, YOU'RE EFFECTIVELY BREACHING MELLANOX LICENSE TERMS, WHICH MIGHT HAVE LEGAL IMPLICATIONS UNDER SOME JURISDICTIONS. IT IS UP TO YOU TO CHECK WHETHER THIS CONSTITUTES AN OFFENCE OR HAS A POTENTIAL OF EXPOSING YOURSELF TO A CIVIL SUIT IN YOUR JURISDICTION AND DECIDE WHAT TO DO ABOUT IT.

My own intentions are scientific/research related only, with no profit generated from it, so I've decided to go ahead with the process and also make information about it public. However, I have also decided not to publish any information that Mellanox might have stronger reservations against, such as exact locations of the MLNX-OS downloadable distros and license keys.

Revision notes

v1.2

- Added Appendix A on how to reduce minimum fan speed on SX6012
 - Credits goes to SGS (STH) for the hex keys and up3up4 (STH) for testing
- Added reference to Appendix A in step 4

v1.1

- Updated by Necrotyr
- Edited formatting
- Removed dead/missing links
- Added references to files that must be replaced
- Added alternate firmware flash procedure from Rand_ (STH)
- Added alternate TFTP command to upload image faster from llowrey (STH)

v1.0

- Initial release by mpogr

About the device

The device we're going to work on has the following components:

1. A Mellanox InfiniBand **board** with varying number of physical ports wired to it. The board is a PCI device that has its own firmware (not to be confused with the software running on the switch CPU). The firmware preinstalled on the non-Mellanox switches is different from the Mellanox one, and this is one of the issues that need to be taken care of during the conversion process. **As a side note, the unmanaged versions of the same switches (the XXX5 models) seemingly have the same board and flashing just the firmware intended for these devices into our managed switch will effectively convert it into such a dumb device. This is the easiest way to make it immediately useful. If you're fine with having a fully functional but unmanaged InfiniBand FDR switch with no Ethernet functionality, you can do just that and stop here.**
2. A **PowerPC CPU** (M460EX). This CPU is big-endian and is used in 32-bit mode.
3. 2GB of **RAM** used by the OS running on the device.
4. 16MB of **Flash EEPROM**. It's mapped into the very end of the addressable RAM space when accessed from the Boot Manager (U-Boot below). It's used for persisting the following 6 distinct entities:

- a. Linux **KERNEL_1**, residing between FF00 0000 and FF1D FFFF. This is the kernel that gets booted when you select the first MLNX-OS instance.
Somewhere inside **KERNEL_1** there is also the **BOOT_1** area (not sure about the exact location, most likely within a well-known reserved area), which is a small hook invoked by the Mellanox version of U-Boot that is responsible for booting the kernel automatically. It also contains the information about the MLNX-OS version it's about to boot.
- b. **KERNEL_2** and **BOOT_2** are same as above but for the second instance of MLNX-OS. Reside between FF20 0000 and FF3D FFFF.
- c. **FDT_1** is so called Flatted Device Tree which is a data structure describing how persistent storage of the device (Flash EEPROM and NAND) is mapped into the character (`/dev/mtdX`) and block (`/dev/mtdblockX`) devices by the Linux kernel. This one is used by **KERNEL_1** and is located between FF1E 0000 and FF1F FFFF.
- d. **FTD_2** is the same for **KERNEL_2**. Located between FF3E 0000 and FF3F FFFF.
FTD_X and **KERNEL_X** have to match, i.e. they have to be extracted from the same MLNX-OS distribution.
- e. **UBOOTENV** is a small area where U-Boot environment variables are kept. They can be manipulated from the U-Boot command prompt and get saved into this area by `saveenv` U-Boot command to persist between boots. Resides between FFF8 0000 and FFF9 FFFF.
- f. **UBOOT** is the device Boot Manager. Resides between FFFA0000 and FFFFFFFF. It gets loaded first when the device is booted.

U-BOOT HAS TO BE HANDLED WITH EXTREME CARE, AS MESSING IT UP WILL BRICK YOUR DEVICE BEYOND REPAIR!!!

It is also important to note that the Mellanox version of U-Boot requires a password to get to its command prompt (it displays a menu first, with the command prompt being one of the options), so, until you're done messing around with your device, don't flash it!

This password can probably be overwritten (not retrieved!), but I haven't tested it yet.

5. 2GB of **NAND storage**. Used for storing the actual MLNX-OS, its configuration and (optional) virtual machines that can be run by it.
6. A **FRU** device with some vendor-specific EEPROM data. Some of the important bits present on the genuine Mellanox switches are either missing or different on our devices, which creates a problem with making MLNX-OS work. We're going to deal with this.
7. 2 1Gbps **Ethernet ports** for the management network connection. On smaller devices (e.g. SX6012) only one of them is wired.
8. 1 **Management Serial Port** wired to an RJ45 socket.

Preparations

You need the following in order to successfully accomplish the conversion:

1. A managed SwitchX/X2 device running software that allows you to get to its U-Boot prompt. The EMC switches are good, because you can easily interrupt their boot sequence by just hitting any key on the serial console.
2. A serial cable wired as RJ45 plug on one end and whatever you have on your PC on the other end (usually a DB9 connector). You absolutely must get it in order to do the job.
3. A pin to reboot the switch (you will have to do it a lot). The ones used to extract SIM cards from mobiles are perfect. Don't use sharp needles!
4. Some Ethernet connectivity to your switch in order to transfer files onto it.
5. Either a TFTP or an NFS server accessible from the switch via its Ethernet connection. TFTP is easier to handle, so I'd suggest to stick to it.
6. A Linux computer for various purposes (extracting files from the distro, manipulating JFFS2 images etc.). Same computer can serve as the TFTP/NFS server above.
7. A current MLNX-OS distribution (version 3.6.1002 as of Aug 5th 2016). Google is your friend.
8. You might also need a license key from Mellanox. I'm a bit fuzzy on that bit at the moment, because of the way I worked with my device. In any case, I'm not going to publish it here for obvious reasons.

Step 0: Connect the switch to a PC via a serial connection

In order to start working on the conversion, you need to connect the switch to a PC via a serial interface (this also works with VMs with the physical serial port passed through). You need to use a terminal program that knows how to handle serial port communications. I use PuTTY. You need to set the comm settings to 9600 bauds.

Once you connect to the switch and hit any button, you will see its output inside your terminal window.

Step 1: Install MLNX-OS Kernel and FDT onto the switch

The vanilla EMC switch doesn't have a Linux kernel preinstalled, so you need to get one yourself first. The best way of doing it is to extract the stock MLNX-OS kernel and FDT from the distro.

The distro comes as an `.img` file, which can be simply unzipped with `unzip`. Inside, you'll find a `.tgz` file that has the entire MLNX-OS distribution tree. The files you need are:

```
/boot/vmlinuz-uni    the kernel
/boot/fdt-uni        the FDT
```

You need to flash them into the correct places from U-Boot. Get into the U-Boot prompt and then (*this is from memory, as I haven't done it in a while, will need to be validated*):

Turn auto-start (e.g. after a `tftp` transfer) off. Not sure what's the difference between the two.

```
setenv autostart off
setenv autoload off
```

Configure the management network interface, so you can properly access the network.

```
setenv ipaddr <youripaddress>
setenv netmask <ournetmask>
```

Save the current environment, so you don't need to do this after every reboot.

```
Saveenv
```

Transfer the MLNX-OS kernel onto the switch and put it in RAM (at address 0x400000).

```
tftp 400000 <yourtftpserverip>:<path>vmlinuz-uni
```

Turn the EEPROM protection off for the relevant area.

```
protect off FF000000 FF1FFFFF
```

Erase the flash, to make sure we wipe it clean just before putting the right stuff there.

```
erase FF000000 FF1FFFFF
```

Copy the contents of the file we've just transferred to the device into the area designated for `KERNEL_1`.

```
cp.b ${fileaddr} FF000000 ${filesize}
```

Now get the FDT.

```
tftp 800000 <yourtftpserverip>:<path>fdt-uni
```

And put it into the right place.

```
cp.b ${fileaddr} FF1E0000 ${filesize}
```

Turn the flash write protection back on.

```
protect on FF000000 FF1FFFFF
```

List the bootable modules in the flash.

```
Imls
```

You should see a bootable image listed now at FF000000 (could be more than one, the other one coming from the EMC OS).

Be extremely careful with the destination addresses of the `protect`, `erase` and `cp.b` commands! If you get these wrong, you can screw up U-Boot and brick your device!

Also, pay attention to the destination addresses (the second argument) of `tftp`. These should be in the RAM range (below FF000000), but not too low, because then `tftp` might overwrite the currently running U-Boot code and it will crash (this is not fatal though, you can just reboot with the pin if this happens). The addresses above (400000 and 800000) should be good in all cases.

You can now boot the kernel and verify it loads OK:

```
run mlxlinux
```

This command actually uses some environment variables defined in the EMC environment, so it's not guaranteed to work. But, if you have a typical EMC switch, it should.

After the kernel loads, it will panic because there is no Linux filesystem on the NAND. This is normal, we're going to fix that.

Step 2: Creating a bootable Linux image for the switch

NOTE: You can skip part of this step if you have downloaded the `files-needed-for-conversion.zip` ZIP-file, it includes the image made in step 2, skip down to the part where we transfer the image to the switch.

In order to proceed further, we need to have a real Linux environment running on the switch. The quickest way of accomplishing this is to create a small Linux package with some essentials, pack it into a JFFS2 image and flash it onto the NAND. Fortunately, we have all the necessary ingredients inside the MLNX-OS distro, so it's quite easy. Make sure you unpack the MLNX-OS distro on the same box that you intend to create the bootable image to prevent any problems with missing permissions. Create an empty directory on your Linux box and, inside it, create the following subdirectory structure:

```
/bin
/dev
/etc
/etc/init.d
/lib
/proc
/sys
/tmp
/var
/var/run
/var/lib
```

Copy `/bin/busybox` from the distro into your `/bin` and create the following symlinks inside it:

```
ln -s busybox bash
ln -s busybox chmod
ln -s busybox cp
ln -s busybox echo
ln -s busybox gunzip
ln -s busybox gzip
ln -s busybox ifconfig
ln -s busybox ln
ln -s busybox ls
ln -s busybox mkdir
ln -s busybox mount
ln -s busybox mv
ln -s busybox rm
ln -s busybox rmdir
ln -s busybox sh
ln -s busybox tftp
ln -s busybox touch
ln -s busybox umount
ln -s busybox vi
```

This next symlink is slightly different to the ones above so please note the path:

```
ln -s busybox ../sbin/init
```

The additional files you need to copy to your `/bin` from the distro are:

```
/bin/tar
/usr/sbin/flash_eraseall (put it also into /bin to simplify things)
```

Inside of your `/etc/init.d` directory create a file named "rcS" with the following contents:

```
mount -t proc none /proc
mount -t sysfs none /sys
```

Then set the correct permissions on the file:

```
chmod a+x /etc/init.d/rcS
```

Now, you need to copy the following files into your `/lib` directory:

```
/lib/ld-2.17.so
/lib/libc-2.17.so
/lib/libcrypt-2.17.so
/lib/libm-2.17.so
/lib/libpthread-2.17.so
/lib/librt-2.17.so
```

and create the following symlinks there:

```
ln -s ld-2.17.so ld.so.1
ln -s ld-2.17.so ld-linux.so.2
ln -s libc-2.17.so libc.so.6
ln -s libcrypt-2.17.so libcrypt.so.1
ln -s libm-2.17.so libm.so.6
ln -s libpthread-2.17.so libpthread.so.0
ln -s librt-2.17.so librt.so.1
```

The last bit here is to create the device files under `/dev`:

```
mknod -m 622 /dev/console c 5 1
mknod -m 666 /dev/null c 1 3
mknod -m 444 /dev/random c 1 8
mknod -m 644 /dev/ttyS0 c 4 64
mknod -m 444 /dev/urandom c 1 9
mknod -m 666 /dev/zero c 1 5
```

It is important to understand that, if we forget something here, it's not the end of the world, as the image we're going to make will have all the instruments to transfer missing files into it should the need arise.

Now, we're going to create a JFFS2 filesystem image from this structure. Here's the command:

```
mkfs.jffs2 --eraseblock=0x20000 --pagesize=2048 --pad --no-cleanmarkers
--big-endian --squash-uids --root=<yourrootdir> --output=<yourimagefilename>
```

Pay attention to the parameters, they are all very important! The erase block and the page size have to match the NAND physical characteristics, and the endianness is important as well (the PPC CPU is big endian).

NOTE: Transfer to switch starts here

Now you're ready to put this image onto the switch. Go back to the U-Boot command prompt and do the following:

```
tftp 400000 <yourtftpserverip>:<yourpath>/<yourjffs2image>
nand erase clean 0 1FFFFFFF
nand write ${fileaddr} 0 ${filesize}
```

It is possible that, while erasing and writing, the NAND will complain about some bad blocks. This is normal and won't create any problem if the amount is not significant (I have 1 in this section of the NAND). They will be safely skipped, so don't worry about it.

At this stage you need to slightly modify the way the MLNX-OS kernel is booted by adding an additional attribute to the kernel command line:

```
setenv image_kernel_args loglevel=2
```

This is because, for some strange and currently inexplicable reason, the JFFS2 images created the way described above throw a lot of ECC errors into the kernel log. This doesn't cause any real trouble, except slowing the booting down to a crawl (because the output to the console is synchronous and is limited by the 9600 bauds serial connection speed), so we need to decrease the log level to make booting of this image faster.

Since we will be modifying the JFFS2 filesystem after it boots, change the write permissions with the following:

```
setenv jffs2_args 'setenv bootargs root=${rootdev} rootfstype=jffs2 rw
reset_button=${reset_button} ${image_kernel_args} ${extra_args}'
```

The only change here is going from ro to rw.

Now we can boot again:

```
run mlxlinux
```

Almost immediately after the kernel finishes loading, you should see an error message and the command prompt. We're in!

You'll get a load of unable to find /dev/tty2-4 errors but since the shell is live you can fire these commands and the errors will stop

```
ln -sf /dev/null /dev/tty2
ln -sf /dev/null /dev/tty3
ln -sf /dev/null /dev/tty4
```

To make life significantly easier for yourself make a symlink to busybox called reboot:

```
ln -s /bin/busybox /bin/reboot
```

Now you don't need to press the reset button every time you need/want to reboot

Step 3: Creating the file systems on the switch and extracting the distro

There is a little bit of housekeeping we need to do first in order to make things work the right way:

```
mount -t proc none /proc
mount -t sysfs none /sys
ifconfig eth0 <youripaddress> netmask <yournetmask>
```

Now you're ready to start making the right filesystem structure. Let's first understand what kind of devices we're dealing with.

There are 2 NAND chips present, each is 1GB of size. Only the first chip is accessible from U-Boot. The FDT that was used during the kernel boot mapped the NAND raw address space into 4 character and block devices as follows:

/dev/mtd6	/dev/mtdblock6	The first MLNX-OS partition, 512MB
/dev/mtd7	/dev/mtdblock7	The second MLNX-OS partition, 512MB
/dev/mtd8	/dev/mtdblock8	The CONFIG area, 100MB
/dev/mtd9	/dev/mtdblock9	The VAR area, 860MB

As our mini-image resides in the first area above, we won't touch it for now, but we'll initialise everything else:

```
flash_eraseall -j /dev/mtd7
flash_eraseall -j /dev/mtd8
flash_eraseall -j /dev/mtd9
```

This can take a bit of time and you also might see messages about some bad blocks, which is, again, normal, as long as the amount is not excessive.

We can now mount these partitions:

```
mkdir -p /mnt/root2
mkdir -p /mnt/config
mkdir -p /mnt/var
mount -t jffs2 /dev/mtdblock7 /mnt/root2
mount -t jffs2 /dev/mtdblock8 /mnt/config
mount -t jffs2 /dev/mtdblock9 /mnt/var
```

Now we can use the available space to transfer the distro `.tgz` file. Let's use temp filesystem for that:

```
mount -t tmpfs tmpfs /mnt/tmp
cd /mnt/tmp
ifconfig eth0 mtu 9000
tftp -b 8192 -gr <imagefullpath> <yourtftpserverip>
```

This will take time, as the image is pretty big and writing to the NAND is quite slow. Once done, we can extract the distro onto the right place:

```
cd /mnt/root2
tar -xvf /mnt/tmp/<imagename>.tgz
```

This will take even longer, just sit there and wait. At the end of the process, we'll have the entire distro contents on the second MLNX-OS partition.

Step 4: Adjusting the MLNX-OS file structure

Now, we need to fix some stuff there before we boot it for the first time.

First, you need to edit the `/etc/fstab` file (you can do it using `vi`, just remember, it is now located under `/mnt/root2/etc/fstab`) to be as follows:

```
/dev/mtdblock7 / jffs2 defaults,noatime,ro 1 1
/dev/mtdblock8 /config jffs2 defaults,noatime 1 2
/dev/mtdblock9 /var jffs2 defaults,noatime 1 2
tmpfs /dev/shm tmpfs defaults 0 0
devpts /dev/pts devpts gid=5,mode=620 0 0
sysfs /sys sysfs defaults 0 0
proc /proc proc defaults 0 0
/dev/cdrom /mnt/cdrom iso9660 noauto,ro 0 0
/dev/fd0 /mnt/floppy auto noauto 0 0
```

The first line is important because it maps the root directory to the second MLNX-OS partition, which is the one we're working on preparing now.

Next, we'll take care of the management Ethernet interface name mapping. The names assigned by the kernel are `eth0` and `eth1`, but some of the utilities expect them to be called `mgmt0` and `mgmt1` respectively. Normally, this is done by some MAC address mapping mechanism happening via the configuration DB (more details below about this one), but I couldn't figure out how this is done exactly. Instead, I just created the `/etc/mactab` file (remember, it's `/mnt/root2/etc/mactab` for us at this stage), and put the following into it

```
mgmt0 XX:XX:XX:XX:XX:XX
mgmt1 YY:YY:YY:YY:YY:YY
```

where `XX:XX:XX:XX:XX:XX` and `YY:YY:YY:YY:YY:YY` are the MAC addresses of `eth0` and `eth1` respectively, which can be retrieved by running `ifconfig -a`.

We also need to adjust the contents of `/etc/rc.d/init.d/rename_ifs`, so it just uses `/etc/mactab` and not whatever it was supposed to do. For that, we open that file with `vi` (again, remembering that it resides under `/mnt/root2/etc/rc.d/init.d/rename_ifs` for us), comment out the line:

```
do_rename_ifs
```

with:

```
# do_rename_ifs
```

and add under it simply:

```
${NAMEIF}
```

Now, a couple of additional touches to the filesystem. First, we need to have the right stuff in `/etc/image_layout.sh`. This file is extremely important during the system initialisation and it needs to have the right stuff in it. It is also used when updating the second MLNX-OS partition, so we want to tackle it. The `image_layout.sh` file can be found in the **files-needed-for-conversion.zip** archive.

Then set the correct permissions:

```
chmod 755 image_layout.sh
```

You will also need to replace the `chad`, `hwd` and `ibd` binaries in your `/mnt/root2/opt/tms/bin` directory with the ones from the **files-needed-for-conversion.zip** archive.

Then set the correct permissions on the files:

```
chmod 755 chad
```

```
chmod 755 hwd
```

```
chmod 755 ibd
```

If you want to change minimum fan speed on an SX6012 now is the perfect time to do it, go to [Appendix A](#) for instructions on how to do it.

Also, there is another script that collides with the one above, and we actually don't need it, so all we need to do is make it empty:

```
rm -f /mnt/root2/etc/layout_settings.sh
```

```
touch /mnt/root2/etc/layout_settings.sh
```

The last script that needs to be modified is `/mnt/root2/etc/customer_rootflop.sh`. We just need to comment out two lines and replace them with set values. These changes are related to the next section where we create an initial manufacturer DB. Comment out the line:

```
# system_profile=`echo $out | grep FEATURE_EN_1: | awk {'print $NF'}`
```

and add under it:

```
system_profile="3"
```

Then comment out:

```
# num_swids=`echo $out | grep FEATURE_EN_10: | awk {'print $NF'}`
```

and add:

```
num_swids="0"
```

The final touch to the root file system:

```
touch /mnt/root2/etc/.firstboot
```

Now, we need to take care of the CONFIG and VAR partitions. First, we can delete the image, it takes a lot space and we don't need it anymore:

```
rm -f /mnt/tmp/<imagename>
```

Now, we can move the entire contents of `/var` and `/config` to where it belongs:

```
mv -f /mnt/root2/var/* /mnt/var
```

```
mv -f /mnt/root2/config/* /mnt/config
```

And the final touch to the `/var` filesystem:

```
touch /mnt/var/opt/tms/.firstmfg
```

Step 5: Creating configuration DB nodes

In this step you will be creating an initial manufacturer DB with some basic values. This needs to be done before your first boot into MLNX-OS. This step should be very straight forward since you will be using the `mddbreq` file in your `/mnt/root2/opt/tms/bin` directory and creating the DB in your config mount. If you encounter any problems with missing libraries when running these commands, then copy them from your extracted image and into your `/lib` directory. Your shared-secret string `<YOURLONGSTRING>` needs to actually be long otherwise you will get errors when trying to boot. I used an 48 character alphanumeric string here. Your chassis serial number can be found in the pull out tab on the front of the switch, if you don't have a serial then use one found on google. It will be changed to the correct one during your first boot anyway.

```
cd /mnt/root2/opt/tms/bin
```

```
mddbreq -c /mnt/config/mfg/mfdb set modify - /mfg/mfdb/cluster/config/enable bool false
```

```
mddbreq -c /mnt/config/mfg/mfdb set modify - /mfg/mfdb/cluster/config/expected_nodes uint16 0
```

```
mddbreq -c /mnt/config/mfg/mfdb set modify - /mfg/mfdb/cluster/config/id string ""
```

```
mddbreq -c /mnt/config/mfg/mfdb set modify - /mfg/mfdb/cluster/config/interface string mgmt0
```

```
mddbreq -c /mnt/config/mfg/mfdb set modify - /mfg/mfdb/cluster/config/shared-secret string <YOURLONGSTRING>
```

```
mddbreq -c /mnt/config/mfg/mfdb set modify - /mfg/mfdb/net/interface/config/mgmt0/addr/ipv4/dhcp bool true
```

```
mddbreq -c /mnt/config/mfg/mfdb set modify - /mfg/mfdb/system/hostid string <YOURSERIALNUMBER>
```

```
mddbreq -c /mnt/config/mfg/mfdb set modify - /mfg/mfdb/system/hostname string <YOURINITIALSWITCHNAME>
```

```
mddbreq -c /mnt/config/mfg/mfdb set modify - /mfg/mfdb/system/hwname string M460EX
```

```
mddbreq -c /mnt/config/mfg/mfdb set modify - /mfg/mfdb/system/layout string MFL1
```

```
mddbreq -c /mnt/config/mfg/mfdb set modify - /mfg/mfdb/system/model string ppc
```

Step 6: Flashing the correct firmware

Step 6.1: Getting the firmware

NOTE: You can skip to step 6.2 if you have an SX6012 or SX6018 and have downloaded the **files-needed-for-conversion.zip** archive, it includes firmware for both switches.

This section will require the use of a hex editor and the .mfa file found in your MLNX-OS image. The exact file name for this version is `/opt/tms/bin/fw-SX-rel-9_3_8170-FIT.mfa`. This is the file we will need to cut apart using a hex editor. I used the website hexed.it to do my editing but I'm sure most hex editing software will work here since the operations we need to do are limited to finding boundaries using headers and cutting out the parts we really need.

First a description of the structure inside the mfa file. It is simply three xz archives combined into one file with a header around them. The structure and headers look like this:

Main Header: <4D 46 41 52 00 00 01 00 00 00 00 00 00 00 01 00 00 01 00 00 09 68>

First XZ Header: <FD 37 7A 58 5A>

Content: Location, Name, Description of Firmwares

First XZ End: <00 00 00 00 01 59 5A>

Second XZ Header: <FD 37 7A 58 5A>

Content: Not Necessary

Second XZ End: <00 00 00 00 01 59 5A>

Third XZ Header: <FD 37 7A 58 5A>

Content: All Firmwares Combined into One File

After unxz-ing the contents

Headers (x96): <4D 54 46 57 8C DF D0 00>

Third XZ End: <00 00 00 00 01 59 5A>

End of File: <A3 2F D0 09>

I used the hex values here since the text equivalents can be interpreted differently depending on the program you are using but they will be easy to spot once you start looking. The first thing you need to do is cut out and save the first and third xz archives inclusive of their headers/footers. In hexed.it this can be done simply by selecting the part you want then right-click and select "Export selected bytes to file". Once you have those two files just unxz them like so:

```
unxz -single-stream <firstfile>.xz
```

```
unxz -single-stream <thirdfile>.xz
```

The first file you will need to read through, either in a text editor or back in hexed.it, to identify the location of your switch model's firmware. The numeric position of this description in the first file should match the numeric location of the actual firmware in the third file. Once you identify that, load the third file into the hex editor and search for the above headers (there are 96 firmwares in this file) then cut out the correct section inclusive of the header and right up to the start of the next header. This new file is your firmware and it will need to be checked then flashed unto the switch. To check that you got the right file run this:

```
flint -i <firmwarename>.bin q
```

Make sure that the PSID matches the one you are expecting!

Step 6.2: Flashing the firmware

Now you have a choice to make on how you want to flash the firmware, do either of the following:

Step 6.2.A: ...using a PC connected to the switch with InfiniBand

You will need your linux box to be hooked up to the switch through infiniband to do an in-band firmware update, so you should have all your drivers and MFT utilities setup there.

First find your switch's id

```
mst status
```

Your switch should be listed and show its `Lid`

Now to burn the firmware into the switch use the following command:

```
flint --allow_psid_change -d lid-0x<yourswitchlid> -i ./<firmwarename>.bin b
```

That's it, at this point your switch is ready for its first boot into your newly setup MLNX-OS partition. Go ahead and reset into the U-Boot environment.

Step 6.2.B: ...from the linux shell on the switch

To flash the firmware locally you need shell access to the running MLNX-OS

Boot into scratch linux (mtdblock6) and adjust admin shell in `passwd` to `/bin/bash` (you will need to remove the symlink of `/mnt/root2/etc/passwd` and create a local copy in `/mnt/root2/etc`).

Transfer created firmware file to the switch using tftp or scp (either in scratch Linux or MLNX-OS)

Boot MLNX-OS by running these commands in U-Boot

```
setenv rootdev /dev/mtdblock7
run mlxlinux
```

Identify target device

```
mst status
```

Make sure one more time you have the correct image

```
flint -i <firmwarename>.bin q
```

flash firmware

```
flint -allow_psid_change -override_cache_replacement -d <device> -i
<firmwarename>.bin b
```

Reboot the switch.

Step 7: First Boot

Once you're back in U-Boot the only thing you need to do here is point rootdev to [/dev/mtdblock7](#) using [setenv](#), don't forget to run [saveenv](#), then [run mlxlinux](#).

You will see the new environment loading up with a lot of new information and possibly a ton of harmless ECC errors. After a few minutes, you will be dropped to the login prompt where you enter with admin/admin. You should get asked to run a basic initial setup and once that's done you should be at the switch CLI prompt and the switch should be reachable through the web as well.

One current issue with the way this is setup is that all of your ports will be running at SDR (2.5Mbps) speed. This is an easy fix though so for now run the following commands from your linux box:

```
ibportstate <yourswitchlid> <port> speed 5
ibportstate <yourswitchlid> <port> espeed 1
ibportstate <yourswitchlid> <port> reset
```

The downside to this approach is that whenever a link is reset you will need to rerun these commands to get the full speed back. Hopefully, a more permanent solution will be found soon.

Everything should be running well at this point but if you need to do any debugging a useful tip is to boot back into your original small jffs boot image and edit the [/mnt/root2/etc/passwd](#) so that the admin user will boot into [/bin/bash](#) and not the CLI. Then break the symbolic links for this file and [/mnt/root2/etc/shadow](#) so that MLNX-OS doesn't overwrite your shell.

Appendix A: Reduce minimum fan speed on SX6012

It is possible to reduce the minimum fan speed from 40% to whatever you're comfortable with by editing the tc file located in /opt/tms/bin/ (/mnt/root2/opt/tms/bin/ in the linux image)

Through experimentation it's been found that the lowest the switch will accept without the fans going crazy is 25% with one PSU and 27% with both PSU's.

Using hexed.it or a similar hex editor search for and modify the **three** instances of this hex value:

```
10 17 F2 7C 00 01 00 00 00 00 00 04 00 00 00 04
00 00 00 04 00 04 00 01 00 05 00 28 00 0A 00 28
00 00 00 63 00 55 FF FF 00 37 FF FF 00 00 00 00
00 03 00 00 00 00
```

I've highlighted the bytes that must be changed, replace them with whatever hex value you want. For reference 28 is equal to 40 and 1B is equal to 27.

Save the file and replace the one from the Mellanox image with this one.

After replacing the file, you need to set the right permissions on it:

```
chmod 755 tc
```