Upgrading Marvell Switch EEPROM and SPI-Flash Version on Line Card

Karthik A Patil and Sowmya K Nag

July 25, 2022

Upgrading Marvell Switch EEPROM and SPI-Flash Version on Line Card

Karthik Patil A
Department of Electronics and Communications
R V College of Engineering
Bengaluru, India
karthikpatila.ec18@rvce.edu.in

Mrs. Sowmya Nag K
Department of Electronics and Communications
R V College of Engineering
Bengaluru, India
sowmyanagk@rvce.edu.in

***Abstract*—In a router, the version of the firmware needs to be upgraded on timely basis. Upgrading the version of Marvell switch EEPROM and SPI-Flash on the line card of the router. If the greater version is available then the current version the upgrade has to be done automatically. In order to auto upgrade the version, there should be a code to detect the available version which is grater then current version. Developing the working script that will successfully upgrade the version of the EEPROM and SPI-Flash. The script should be able to read the current version from the system and can able to compare it with available version. Before that, working script that is used for upgrading the version should be built, the system file needs to be updated with name and specifications of the memory units, tag values are assigned to the EEPROM and SPI-flash.The upgrade file, on which code for version upgrade written is tested on hardware lab devices and its verified that version of the EEPROM and SPI-flash has upgraded and it should reflect when checked for system version.In this whole process there are changes has to be brought into related files so that these files helps auto upgrading the version on CLI. If the unit testing of the code is passed then its further proceeded for testing code through CLI. After successful testing and validation finally the script needs to be committed for reviewing. The outcome of the PCT are, the committing speed is increased for EVO about 5% and decreased about 4% to Junos. The toxic PR of the Junos has been resolved. The multi touch commits are increased about 2% fot EVO and about 10% for Junos. The build infra failure rate has been decreased about 1% to EVO and 6.0% for Junos. Due to changes made the sandbox buildtime has increased. The test time decreased for EVO about 0.6% and increased for Junos about 0.4%.**

***Index Terms*—EEPROM - Electrically Erasable Programmable Read only Memory, SPI-flash - Serial Peripheral Interface Flash, CLI - Command Line Interface**

I. Introduction

Software versioning is a way to categorize the unique states of computer software as it is developed and released. The version identifier is usually a word, a number, or both. Flash memory is an Electronically Erasable and Re-Programmable memory chip. The Flash memory contains the full Operating System Image (IOS, Internetwork Operating System). This allows you to upgrade the OS without removing chips. Flash memory retains content when router is powered down or restarted. Routers use flash memory, rather than disks, for storing information. Flash storage media is significantly more expensive and slower than disk storage, but the amount of storage needed to run a router is relatively small compared to the amount needed to run a general-purpose computer. Flash also has the important benefit that it tends to be more reliable than disk storage.

EEPROM which is Electrically Erasable Programmable Read-Only Memory, is a type of memory in which data is read, written, and erased at the byte level. Flash memory, on the other hand, which is a type of EEPROM, is architecturally arranged in blocks where data is erased at the block level and can be read or written at the byte level. SPI flash memory and EEPROMs are both considered non-volatile memory. Non-volatile memory means that the device is able to retain data without requiring a constant power supply, allowing devices to store information even when powered off. They are both electronically writable and erasable memories and are microcontroller-based applications, meaning they are used either on-chip or off-chip to store information.

While flash memory and EEPROM devices are both capable of storing information used in embedded devices, their architecture and operations for reading, writing and erasing data differ slightly. SPI flash memory, also known as flash memory, has become widely used in the embedded industry and is commonly used for storage and data transfer in portable devices. Common devices include phones, tablets, and media players, as well as industrial devices such as security systems and medical products. Flash memory is particularly useful for static data applications such as USB flash drives.

II. Background

The upgradation process has taken manually before developing the script for firmware upgradation which will upgrade the firmware of the ethernet switches automatically. Where the new version image is built and one of our Engineer should visit the site and perform the upgradation of the firmware manually. This process has to take weeks, time and cost are wasted to do the same. And the router will be unavailable until the upgradation finish. Now, the whole upgradation process will complete just in minutes and with one CLI command. Developing the code which will upgrade the firmware of memory elements like EEPROM and SPI-Flash to the latest available version. Testing the script in the hardware lab devices. Verifying the results and getting approval from the review team. Achieving successful system testing for implementation of the script.

The main objectives of the project are: To build a sandbox of latest version on both Junos OS and EVO OS. Adding required repositories where files that need to be changed are present. Changes are brought on respective repositories. To build a working script that will upgrade EEPROM and SPI-Flash to the higher version. Additional script are written to support upgrade. Building the final image on Junos OS and EVO OS for testing on RE CLI. To unit test the script and analysing the results. Testing complete modification on the CLI. Getting approval form the review team. Committing the script for system testing and final shipping for final implementation.

III. Implementation

To develop a code and test its working, first one should build an environment where the code can be written and tested. Here the environment is BaaS server and Linux Operating system. Creating sandbox which will work on Junos and EVO operating system of latest version. Developing the code for upgrading the firmware version of the EEPROM and Serial Peripheral Interface Flash (SPI-FLASH). Adding repositories to the sandbox where the script files needs to be updated to support the version upgradation. And these changes are brought on both Junos and EVO OS. After the script is developed and changes are updated, the images consisting of all the modifications are built independently on junos and EVO sandbox which will be subjected for unit testing on router. Verifying the results of unit test and further testing script on Command Line Interface (CLI) of the router engine. Once the testing is successful, code is submitted for reviewing and review team will verify the changes and approve it for further commit.The code will be implemented after passing the system testing. The unit test results and the steps that are used to upgrade are sent to systest team. And script will be approved for implementation

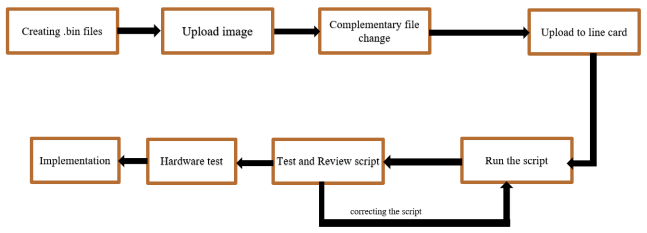


Fig. 1. Design Methodology

Build as a Service (BaaS) is a next generation container-based virtual build envi- ronment for development-based work-flows. It is more of an on-demand workload driven environment rather than a static VM. This means that hardware resources are allocated to you when in use and they are returned when not in use. It uses a high-end storage solution with container on Kubernetes.

A. Creating Volume and Sandbox

To checkout code from the source code repository, one need to create a sandbox. One can either create a new sandbox or checkout a prebuild sandbox. The prebuild sandbox contains precompiled objects and therefore, the builds will be faster. To create a new sandbox in the volume, run a traditional mksb command if you are checking out code from the Junos-2009 repository. This will take about 10 minutes.

As prebuild sandboxes are already built and available on the cloud. One need to first checkout a prebuild sandbox to your sandbox. This will clone volume from the prebuild sandbox to your volume. As prebuild sandboxes are pre-compiled objects, it makes builds faster and also ensures that the code that one checked out will build without errors. Building means compiling the code. The baas build command is used to create a build.

B. Code Design

The code is written on the specific repositories where on can access the code. The script for version upgrade is written on the file *bugatti_switch_upgrade.sh* this file is built on specific repository called *platform_utiles* on which this script file can be accessed. On Evo sandbox script for upgrade is developed, the code for compare the current version and available version is build on capdb repository. On hardware d repository the code is built to call the files for get the current version and check the available version. It also pass the parameters for those files which are called to get the version. This file has the greater usage when it comes to auto upgrade. All the changes which are brought on these repositories will supports the auto upgrade the firmware version. Because once the chassis handled over daemon, manually passing of the parameters won't be possible.

The code required for upgrade the EEPROM and SPIFlash version using the bin files should be developed in such a way that it should copy the bin files from the router standard folder to the present working path. The data has to be written byte by byte to the EEPROM and can be written in bulk for flash. Since, there are available script that will do the upgrade can be called here in the script so the upgrade is done. The specific parameters should be passed to the script to upgrade particular units like EEPROM and SPIFlash these are ethernet switches and upgraded separately by calling *bugatti_switch_upgrade.sh* file. The code is written in such a way that the current version can be accessed using the same file only by passing the different parameters. The parameters passed to check the verion are *"FRU","version", "eeprom/spiflash"* and for upgrade the parameters should be passed are *"FRU", "upgrade", "eeprom/spiflash", "path to image files"*. The data of the image file is shown in Figure 3.1. Meanwhile size of EEPROM .bin image file is 6kb and size of the SPIFlash file is ablout 683kb. Where these files are passed with upgrade script for version upgrade.

Same parameters will be present in fdt.h file. Which will take care of passing parameters and calling *bugatti_switch_upgrade.sh* file with these parameters. Which will helps the upgradation when daemon are taken it. On Junos sandbox there *jfirmware_data* file needs to be updated so that the router will detect the EEPROM and

SPIFlash and their specifications like available version which is taken from here. And EEPROM and SPIFlash systems will be reflected when user whats to see the system present in router along with their specifications.

C. *Final Image Building*

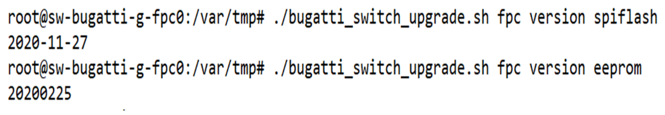
Once the complete code is developed and script are modified for support the upgrade. The independent images are built on junnos and evo platforms. tese images are consisting of all the changes and are zipped in .tgz fromate which should be unzipped on the router. The sample of the image from junos are *junos-vmhost-install-mx-x86-64-22.3I20220526_0838_kpatila.tgz*, and for evo side is *junos-evo-install-ulc-mx-x86-64-22.3I20220526102452-EVO_kpatila.tgz*. These images are copied onto the router for testing the changes. Good image must be build in order to proper working on the router or else the router components will crash and it will be recovered manually. These images are built to test the code change for CLI testing.

IV. Testing the Script

A. *Unit testing*

The main objective of unit testing is to isolate written code to test and determine if it works as intended. Unit testing is an important step in the development process, because if done correctly, it can help detect early flaws in code which may be more difficult to find in later testing stages.

A unit test typically comprises of three stages: plan, cases and scripting and the unit test itself. In the first step, the unit test is prepared and reviewed. The next step is for the test cases and scripts to be made, then the code is tested. Test-driven development requires that developers first write failing unit tests. Then they write code and refactor the application until the test passes.

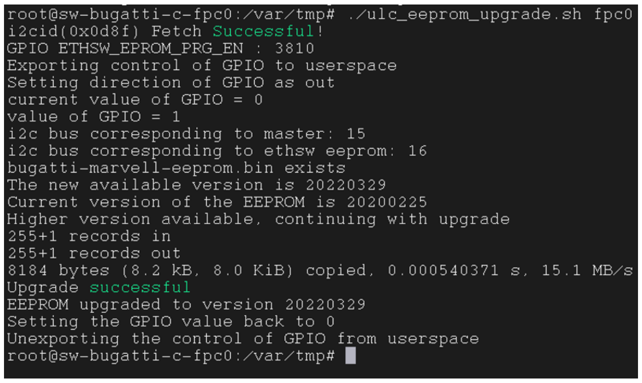


```
root@sw-bugatti-g-fpc0:/var/tmp# ./bugatti_switch_upgrade.sh fpc version spiflash
2020-11-27
root@sw-bugatti-g-fpc0:/var/tmp# ./bugatti_switch_upgrade.sh fpc version eeprom
20200225
```

Fig. 2. Versions before upgrading

Here the *bugatti_switch_upgrade.sh* file along with .bin image files of EEPROM and SPIFlash are tested manually and the errors are debugged. The code is written in a way if the image files are not found on the line card it is copied from the router directory where it is available. And the parameters are also passed manually to test the upgrade. Debugging the errors and developing the complete working script are main objectives of the unit testing. Here in this section the version upgrade is tested.

Taking note of the versions of the EEPROM and SPI-FLASH before upgrade. The Figure 2. shows the version of EEPROM and SPI-FLASH before upgrading. The version of EEPROM is 20200225 and 2020-11-27. In which where the major and minor version are year and month, so the version that will printed on the CLI console are 2020.2.0 for EEPROM and 2020.11.0 for SPIFlash. Figure 2 shows the current version of the ethernet switch's before upgrading.



```
root@sw-bugatti-c-fpc0:/var/tmp# ./ulc_eeprom_upgrade.sh fpc0
i2cid(0x0d8f) Fetch Successful!
GPIO ETHSW_EPROM_PRG_EN : 3810
Exporting control of GPIO to userspace
Setting direction of GPIO as out
current value of GPIO = 0
value of GPIO = 1
i2c bus corresponding to master: 15
i2c bus corresponding to ethsw eeprom: 16
bugatti-marvell-eeprom.bin exists
The new available version is 20220329
Current version of the EEPROM is 20200225
Higher version available, continuing with upgrade
255+1 records in
255+1 records out
8184 bytes (8.2 kB, 8.0 KiB) copied, 0.000540371 s, 15.1 MB/s
Upgrade successful
EEPROM upgraded to version 20220329
Setting the GPIO value back to 0
Unexporting the control of GPIO from userspace
root@sw-bugatti-c-fpc0:/var/tmp# █
```

Fig. 3. EEPROM version Upgrading

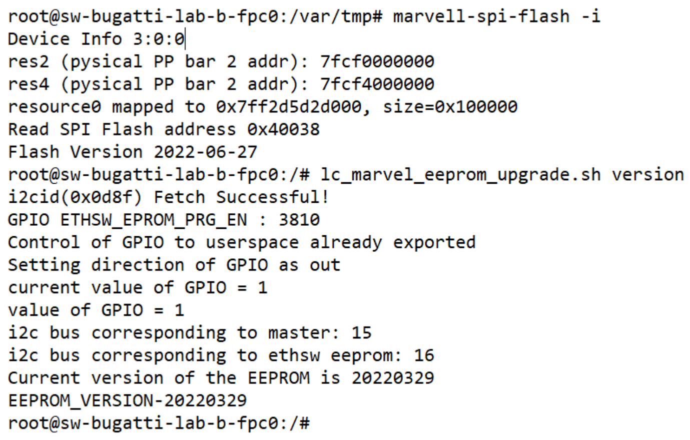
In the Figure 3, EEPROM upgradation is processed. The upgradation file along with suitable parameters are passed for upgrading EEPROM. The data erased and rewritten byte by byte and not as whole block. The code will check for the image bin file which is necessary for version upgrade, and the i2c master and slave busses are selected. Here the available version is 20220229 where current version is 20200225. The upgrade will only continue if the available version is higher than current version. After the upgrade is complete the upgrade successful message will be printed. The GPIO pins are toggled before upgrading the GPIO pins are set to high, so that the data can be written on EEPROM. After upgrade the GPIO pin will be value back to 0. So that data cannot be interpreted on the EEPROM.

B. *Unit Testing Results*

The SPIFlash can be upgraded by executing the main file *marvell-spi-flash* with the upgrading image file as a parameter. Here, the current version of the SPIFlash is erased and the version that is passed while executing the command are programmed. After the completion SPIFlash upgrade, one should check that the line card should reboot properly. The new version will be processed only after one should do power cycle, that is line card should be turned off and then on. Then the new version will be accepted. While doing power cycle if there is any anomaly in the script then the old version will be there are backup it will erase the upgraded version by installing old version.

For EEPROM upgrade is shown in Figure 2, the size of the image file is 8184 bytes where its written by taking 255 bytes at a time. In upgrading the EEPROM only upgrading version to the higher version is only possible, the script is built in such a way that downgrading the version is not possible. Once the EEPROM is upgraded it cannot be downgraded using this script.

After the Unit Testing of the code is successfully the changes and script are tested in CLI. The entire changes are taken note. The auto upgradation is tested.



```
root@sw-bugatti-lab-b-fpc0:/var/tmp# marvell-spi-flash -i
Device Info 3:0:0
res2 (pysical PP bar 2 addr): 7fcf0000000
res4 (pysical PP bar 2 addr): 7fcf4000000
resource0 mapped to 0x7ff2d5d2d000, size=0x100000
Read SPI Flash address 0x40038
Flash Version 2022-06-27
root@sw-bugatti-lab-b-fpc0:/# lc_marvel_eeprom_upgrade.sh version
i2cid(0x0d8f) Fetch Successful!
GPIO ETHSW_EPROM_PRG_EN : 3810
Control of GPIO to userspace already exported
Setting direction of GPIO as out
current value of GPIO = 1
value of GPIO = 1
i2c bus corresponding to master: 15
i2c bus corresponding to ethsw eeprom: 16
Current version of the EEPROM is 20220329
EEPROM_VERSION-20220329
root@sw-bugatti-lab-b-fpc0:/#
```

Fig. 4. Upgraded Version of the Ethernet switches

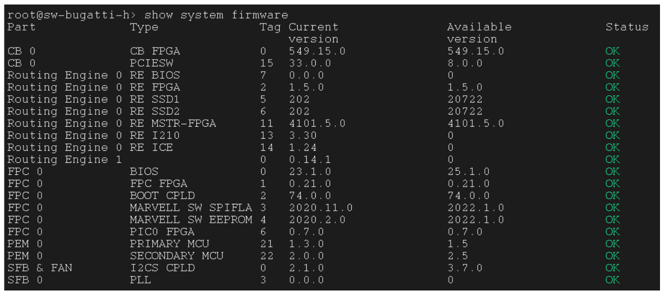
Figure 4, shows that the version of the EEPROM and SPIFLASH firmware version after the upgradation. To find the current version of the Ethernet switch like EEPROM and SPIFLASH execute *bugatti_switch_upgrade.sh* file along with the parameters *"FRU", "version", "eeprom/spiflash"* the version will be printed on the console. Another method is to execute the command *marvell-spi-flash -i* which will give the Flash version. Here, the Flash version is 2022-06-27. Meanwhile, for EEPROM also there are generic script where like *lc_marvel_eeprom_upgrade.sh* executing this script along with parameter *"version"* it will give the EEPROM version which is 20220329. By this the UT test results are verified that the upgrade script *bugatti_switch_upgrade.sh* works correctly. And now the complementary changes that are made to support the auto upgradation of the script are tested on CLI of the router engine.

C. CLI Testing

The daemon will handle every procedure, where the upgradation also taken care by daemon. So the auto upgradation should be tested and verified before the changes made are implemented. For CLI testing the .tgz image should be built independently on junos and evo platform.

The images will look like *junos-vmhost-install-mx-x86-64-22.3I20220527_0829_kpatila.tgz* for junos and *junos-evo-install-ulc-mx-x86-64-22.3I20220527111852-EVO_kpatila.tgz* for evo side. The changes that are made in junos and evo are captured in those image. The junos vmhost image are installed onto the router for support the working condition and to apply the changes on the junos side. It will take some time to install the junos image and reboot the router. The changes are visible in the directory on the router. Mainly checking that the SPIFlash and EEPROM changes in *jfirmware_data* file where the current and available version tag value and the image file to upgrade are modified. The available version is taken from *jfirmware_data*.

Once the evo image is updated on the line cards, the changes that are made in *jfirmware_data* file are now visible on the

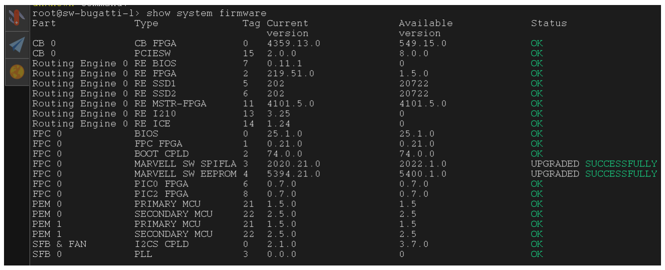


```
root@sw-bugatti-h> show system firmware
Part               Type               Tag Current                Available              Status
                                          version                version
CB 0               CB FPGA            0   549.15.0               549.15.0               OK
CB 0               PCIESW             15  33.0.0                 8.0.0                  OK
Routing Engine 0 RE BIOS              7   0.0.0                  0                      OK
Routing Engine 0 RE FPGA              2   1.5.0                  1.5.0                  OK
Routing Engine 0 RE SSD1              5   202                    20722                  OK
Routing Engine 0 RE SSD2              6   202                    20722                  OK
Routing Engine 0 RE MSTR-FPGA         11  4101.5.0               4101.5.0               OK
Routing Engine 0 RE I210              13  3.30                   0                      OK
Routing Engine 0 RE ICE               14  1.24                   0                      OK
Routing Engine 1                      0   0.14.1                 0                      OK
FPC 0              BIOS               0   23.1.0                 25.1.0                 OK
FPC 0              FPC FPGA           1   0.21.0                 0.21.0                 OK
FPC 0              BOOT CPLD          2   74.0.0                 74.0.0                 OK
FPC 0              MARVELL SW SPIFLA 3    2020.11.0              2022.1.0               OK
FPC 0              MARVELL SW EEPROM 4    2020.2.0               2022.1.0               OK
FPC 0              PIC0 FPGA          6   0.7.0                  0.7.0                  OK
PEM 0              PRIMARY MCU        21  1.3.0                  1.5                    OK
PEM 0              SECONDARY MCU      22  2.0.0                  2.5                    OK
SFB & FAN          I2CS CPLD          0   2.1.0                  3.7.0                  OK
SFB 0              PLL                3   0.0.0                  0                      OK
```

Fig. 5. CLI console output after updating evo image

CLI console. Figure 5, which show that the changes that are brought. The MARVELL SW EEPROM and MARVELL SW SPIFLASH which are present on FPC 0 linecard along which that the tag value, current version, available version and the status of the system units are printed on the console. One can see that current version of the SPIFlash and EEPROM are 2020.11.0 and 2020.2.0 respectively. Available version are 2022.1.0 and 2022.1.0 for both units. These available version are modified manually during the time of upgrade in *jfirmware_data* file.

The upgrade will only happen when the available version is greater than current version during auto upgrade or CLI upgrade. If the available version is equal or lower than current version the upgrade won't happen it will give daemon timeout error, where the daemon could not able proceed for upgrade.



```
root@sw-bugatti-l> show system firmware
Part               Type               Tag Current                Available              Status
                                          version                version
CB 0               CB FPGA            0   4359.13.0              549.15.0               OK
CB 0               PCIESW             15  2.0.0                  8.0.0                  OK
Routing Engine 0 RE BIOS              7   0.11.1                 0                      OK
Routing Engine 0 RE FPGA              2   219.51.0               1.5.0                  OK
Routing Engine 0 RE SSD1              5   202                    20722                  OK
Routing Engine 0 RE SSD2              6   202                    20722                  OK
Routing Engine 0 RE MSTR-FPGA         11  4101.5.0               4101.5.0               OK
Routing Engine 0 RE I210              13  3.25                   0                      OK
Routing Engine 0 RE ICE               14  1.24                   0                      OK
FPC 0              BIOS               0   25.1.0                 25.1.0                 OK
FPC 0              FPC FPGA           1   0.21.0                 0.21.0                 OK
FPC 0              BOOT CPLD          2   74.0.0                 74.0.0                 OK
FPC 0              MARVELL SW SPIFLA 3    2020.21.0              2022.1.0               UPGRADED SUCCESSFULLY
FPC 0              MARVELL SW EEPROM 4    5394.21.0              5400.1.0               UPGRADED SUCCESSFULLY
FPC 0              PIC0 FPGA          6   0.7.0                  0.7.0                  OK
FPC 0              PIC2 FPGA          8   0.7.0                  0.7.0                  OK
PEM 0              PRIMARY MCU        21  1.5.0                  1.5                    OK
PEM 0              SECONDARY MCU      22  2.5.0                  2.5                    OK
PEM 1              PRIMARY MCU        21  1.5.0                  1.5                    OK
PEM 1              SECONDARY MCU      22  2.5.0                  2.5                    OK
SFB & FAN          I2CS CPLD          0   2.1.0                  3.7.0                  OK
SFB 0              PLL                3   0.0.0                  0                      OK
```

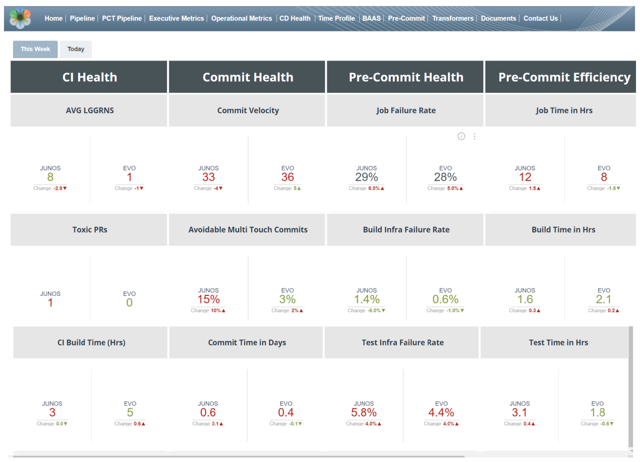
Fig. 6. CLI console output after upgrading

After the user executes the command *"request system firmware upgrade fpc slot 0"* command on router engine CLI, which will starts checking every component present on FPC 0 line card for version available for upgrade. And it will proceed for upgrade if the upgrade is available. The status will change according to the outcome. The status will change from "OK" to "UPGRADED SUCCESSFULLY" for successful upgrade and "UPGRADE FAILED" if any errors occurred while upgrading.

V. Results

The results of the firmware upgradtion will be available after the Pre commit tool completes its job. The Job consists of complete testing of the script, adaptability of the script and the effects caused be script on other components of the router. From the Figure 7, shows the outcome of the pre commit tool. Where the committing speed is increased for EVO about 5%

and decreased about 4% to Junos. The toxic PR of the Junos has been resolved. The multi touch commits are increased about 2% fot EVO and about 10% for Junos. The build infra failure rate has been decreased about 1% to EVO and 6.0% for Junos. Due to changes made the sandbox buildtime has increased. The test time decreased for EVO about 0.6% and increased for Junos about 0.4%. The overall result is the script for ethernet switch upgradation is accepted and shipped in upcoming releases to MX304 router.


Home | Pipeline | PCT Pipeline | Executive Metrics | Operational Metrics | CD Health | Time Profile | BAAS | Pre-Commit | Transformers | Documents | Contact Us |
This Week
Today

CI Health		Commit Health		Pre-Commit Health		Pre-Commit Efficiency	
AVG LGGRNS		Commit Velocity		Job Failure Rate		Job Time in Hrs	
JUNOS 8 Change -2.9▼	EVO 1 Change -1▼	JUNOS 33 Change -4▼	EVO 36 Change 5▲	JUNOS 29% Change 6.0%▲	EVO 28% Change 5.0%▲	JUNOS 12 Change 1.5▲	EVO 8 Change -1.8▼
Toxic PRs		Avoidable Multi Touch Commits		Build Infra Failure Rate		Build Time in Hrs	
JUNOS 1	EVO 0	JUNOS 15% Change 10%▲	EVO 3% Change 2%▲	JUNOS 1.4% Change -6.0%▼	EVO 0.6% Change -1.0%▼	JUNOS 1.6 Change 0.3▲	EVO 2.1 Change 0.2▲
CI Build Time (Hrs)		Commit Time in Days		Test Infra Failure Rate		Test Time in Hrs	
JUNOS 3 Change 0.0▼	EVO 5 Change 0.6▲	JUNOS 0.6 Change 0.1▲	EVO 0.4 Change -0.1▼	JUNOS 5.8% Change 4.0%▲	EVO 4.4% Change 4.0%▲	JUNOS 3.1 Change 0.4▲	EVO 1.8 Change -0.6▼

Fig. 7. Pre Commit Tool Results

VI. Conclusion

In most cases, EEPROM was used to allow software upgrades to installed devices. EEPROM download was generally accomplished through a local serial port. A CPU is needed, along with memory for storage of both the management code and the MIB data structures. The code store is usually implemented in nonvolatile, memory; EEPROM or Flash ROM is typically used. This allows the management capability to be available upon device initialization, without the need for either code download or a local mass storage device (i.e. disk), while still allowing code modifications, bug fixes, and minor upgrades without a hardware change.

The main objective is to build a suitable script to upgrade the Mravell Switch EEPROM and SPIFlash. And make the switch visible on CLI console window. For that, the separate script is built for upgrading. So that the firmware upgradation will be done with one command. The changes and modification are brought in linked files which will support the auto upgradation. And the code is tested in various steps like Unit Test, CLI testing, verification and Pre Commit Testing. After the code successfully tested and the results are validated the code will be released for customer. The EEPROM and SPIFlash firmware auto upgradation script has proven to be efficient in saving time and resources.

References

[1] M. Ziehensack and M. Kunz, "Smart ethernet switch architecture," IEEE Standards Asso. Ethernet IP@ Automot. Technol. Day, 2017.

[2] D. Valencic, "Vendors' implementation of netconf standard on routers and switches," in 2020 43rd International Convention on Information, Communication and Elec- tronic Technology (MIPRO), IEEE, 2020, pp. 536–541.

[3] A. Astarloa, J. L azaro, U. Bidarte, J. A. Araujo, and N. Moreira, "Fpga imple- mented cut-through vs store-and-forward switches for reliable ethernet networks," in Design of Circuits and Integrated Systems, IEEE, 2014, pp. 1–6.

[4] T. Docquier, Y.-Q. Song, V. Chevrier, L. Pontnau, and A. Ahmed-Nacer, "Deter- mining a tight worst-case delay of switched ethernet network in iec 61850 architec- tures," in 2020 IEEE 45th Conference on Local Computer Networks (LCN), IEEE, 2020, pp. 184–194.

[5] M. Davies, A. Lines, J. Dama, et al., "A 72-port 10g ethernet switch/router us- ing quasi-delay-insensitive asynchronous design," in 2014 20th IEEE International Symposium on Asynchronous Circuits and Systems, IEEE, 2014, pp. 103–104.

[6] K. Solanki et al., "Design of efficient noc router for chip multiprocessor," in 2016 International Conference on Inventive Computation Technologies (ICICT), IEEE, vol. 3, 2016, pp. 1–4.

[7] M. P. Daf and B. B. Saynkar, "Performance and evaluation of loopback virtual channel router with heterogeneous router for on-chip network," in 2014 Fourth International Conference on Communication Systems and Network Technologies, IEEE, 2014, pp. 1065–1069.

[8] B. Ramanathan, "Deadlock free hardware router with dynamic arbiter," in 2018 8th International Conference on Intelligent Systems, Modelling and Simulation (ISMS), IEEE, 2018, pp. 110–113.

[9] B. Dec and A. Pfitzner, "Feasibility studies of eeprom memory implementations in vestic technology," in 2018 25th International Conference" Mixed Design of Inte- grated Circuits and System"(MIXDES), IEEE, 2018, pp. 275–279.

[10] V. R. Banala, C. Hao, and C. Hutchens, "Secure interface architecture for charge trap transistor (ctt) based eeprom," in 2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS), IEEE, 2019, pp. 219–222.

[11] A. Pervaiz, M. Younas, A. G. Hashmi, and H. W. Malik, "An economical distributed design of a hardware based router," in Student Conference On Engineering, Sciences and Technology, IEEE, 2004, pp. 86–92.

[12] Y. Sekiyama, Y. Fujihara, T. Hayashi, et al., "Timing-oriented routers for pcb layout design of high-performance computers," in 1991 IEEE International Con- ference on Computer-Aided Design Digest of Technical Papers, IEEE Computer Society, 1991, pp. 332–333.

[13] T. Lee and Y.-Y. Huang, "Analysis and synthesis techniques of router circuits," in 2015 4th International Conference on Computer Science and Network Technology (ICCSNT), IEEE, vol. 1, 2015, pp. 838–841.

[14] A. A. Mulajkar, S. K. Sinha, and G. S. Patel, "Tcmp and chipper router design for power efficient network on chips," in 2021 International Conference on Computer Communication and Informatics (ICCCI), IEEE, 2021, pp. 1–4.

[15] W. Hou, L. Guo, Q. Cai, and L. Zhu, "3d torus onoc: Topology design, router modeling and adaptive routing algorithm," in 2014 13th International Conference on Optical Communications and Networks (ICOCN), IEEE, 2014, pp. 1–4.

[16] X. Zhao, G. Shen, W. Shao, and S. K. Bose, "Energy efficient and bandwidth guaranteed design for optical network with mixed sleep-enabled and non-sleep- enabled router cards," Journal of Lightwave Technology, vol. 34, no. 4, pp. 1072– 1085, 2015.

[17] C. H. Ng, "A symbolic-interconnect router for custom ic design," in 21st Design Automation Conference Proceedings, IEEE, 1984, pp. 52–58.

[18] T. Datta and C. Muralidharan, "Definition, design development of the ixe2424 network switch/router asic," in Proceedings of ASP-DAC/VLSI Design 2002. 7th Asia and South Pacific Design Automation Conference and 15h International Con- ference on VLSI Design, IEEE, 2002, p. 801

[19] P. Yao, H. Wang, Y. Liu, J. Niu, Z. Zhu, and L. Lin, "Integrated railway smart grid architecture based on energy routers," Chinese Journal of Electrical Engineering, vol. 7, no. 4, pp. 93–106, 2021.

[20] M. D. Moffitt, "Maizerouter: Engineering an effective global router," IEEE Transac- tions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 11, pp. 2017–2026, 2008.

[21] A Kamaraj, S Ramya, et al., "Design of router using reversible logic in quantum cel- lular automata," in 2014 International Conference on Communication and Network Technologies, IEEE, 2014, pp. 249–253