White Paper

Intel® Open FPGA Stack

Intel® FPGA Linux Community Out-of-Tree Device Driver

Authors

Tom Rix
Hardware Accelerators Product Owner
Red Hat

Rony Schutz
Product Marketing Specialist
Intel Programmable Solutions Group

Tamara Lin
Product Marketing Specialist
Intel Programmable Solutions Group

Abstract

This white paper is intended for kernel developers who need to quickly develop an installable driver for Intel's family of FPGA programmable acceleration cards (PACs), Acceleration Development Platforms (ADPs), SmartNICs, and third-party Intel-FPGA based acceleration solutions. Development will leverage a community supported out of tree backport driver. Instructions are provided on how to add the minimum amount of code to create a testable rpm.

Background

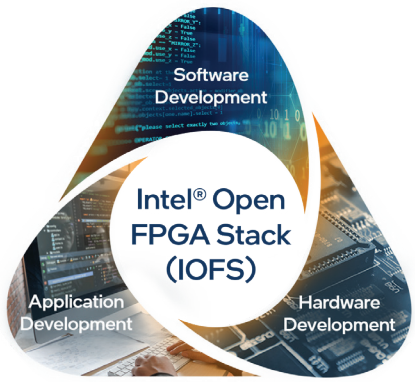
Upstreaming source code is the process of submitting changes to existing open-source projects to become integrated in the overall mainline (or upstreamed) codebase. The upstreaming initiative offers a wide range of benefits for new products in development—from community code reviews and contributions to lifetime maintenance and reusability with API changes. Upstreaming to the Linux kernel also ensures that your source code can be reused by every Linux distribution, a vital proof point for selling your hardware to existing Linux users.

Integrating source code in the mainline Linux kernel is a formal process that takes time and persistence. The process of upstreaming requires multiple rounds of vetting and inspection to ensure the code being merged is the highest quality for the maintainers. While upstreaming drivers to the Linux kernel is often an ideal solution for developers and customers, their product timelines, resources, or bandwidth may require an intermediate solution that avoids the longer upstreaming cycle. In such instances where customers don't need immediate native support from their operating system (OS) vendor or to bring-up/test new hardware on a production Linux distribution, backporting drivers is a viable early enablement solution.

Table of Contents

Intel OFS

One such case study is with Intel® Open FPGA Stack, or Intel® OFS, a new hardware and software technology from Intel enabling users to customize unique, FPGA-based acceleration platform solutions. Intel OFS provides the infrastructure, design, and software components needed to reduce development time by delivering code that can be used as-is or modified to meet specific product requirements.



Abstract

Intel OFS is being developed with an open-source methodology, delivering hardware and software source code through git repositories. Hardware source code provided through git repositories instantiates the core FPGA infrastructure and interfaces, static and dynamic partial reconfiguration regions, and a board management controller. Software source code managing the FPGA is also provided through git repositories, including a lightweight user-space library, user-space software tools, and kernel space Linux drivers. Intel OFS also uses a Device Feature List (DFL) structure to abstract the FPGA contents to software.

Intel has been working to upstream their Intel OFS Linux kernel drivers to the mainline kernel with the goal of developing a completely open-source project sustained by the developer community. By upstreaming Intel's board kernel drivers, the goal is for software vendors like Red Hat to provide native support for Intel OFS reference platforms utilizing Intel® Stratix® 10 and Intel® Agilex™ FPGAs. In the interim of upstreaming kernel drivers, Intel has developed a community backport driver in collaboration with Red Hat.

Development Considerations

While the kernel driver can be developed on any kernel repository, register transfer level (RTL) tools and software acceleration stacks depend on not just specific kernel versions, but also specific linux distribution versions. To upstream the driver, it must be developed against the unstable linux-next branch. Linux-next becomes the mainline linux kernel approximately every 3 months, and a linux distribution will synchronize with the mainline every 3-12 months. RTL tools and acceleration stacks require an additional 3-12 months after that. The lag generated by this process prevents developers from immediately getting started with new hardware and means they lose a competitive edge in time to market. The interim solution would be for the hardware vendor to release an out-of-tree kernel driver to cover this initial development period. However, that may prove to be too daunting of a task from individual vendors or developers. Instead, developers should simply join a community supported driver project so they only need to add their new, unique hardware or features to an existing project. The next sections will demonstrate how to port a development kernel to a deliverable out-of-tree kernel driver package.

Linux DFL

Repository location: https://github.com/OPAE/linux-dfl

Linux DFL (device feature list) is the linux kernel development repository for Intel FPGA acceleration boards that use the device feature list to discover and enumerate a board's intellectual property (IP) blocks. The DFL contains the new features and boards that are in the process of being upstreamed to the mainline kernel.

There are several development branches, however the out of tree driver is based on stable branches. The Linux DFL stable branches follow the main Linux kernel long-term branches at git.kernel.org (Linux stable kernel). At the time of writing this, the latest long-term branch is 5.10.y.

5.10.y branch location: https://github.com/OPAE/linux-dfl/tree/fpga-ofs-dev-5.10-lts

These branches all have a -lts (Long Term Support) prefix. They are kept current by regular merging with their upstream. For example, this is a merge from 5.10.78 to 5.10.81 https://github.com/OPAE/linux-dfl/commit/30bf5bb27262f706321353e0e6361d71802ecd24

The supported *-lts branch is always the latest long-term branch. When a new long-term branch is released, a new *-lts branch is created to follow it. This translates into a lifetime of about a year.

Because the out-of-tree driver cherry picks commits from Linux DFL, all the real changes should be in Linux DFL. Changes are accepted in the usual GitHub manner, through pull requests.

Linux DFL Backport

Repository location: https://github.com/OPAE/linux-dfl-backport

The linux-dfl-backport driver is an evolution of the earlier opae-intel-fpga-driver that was packaged with some versions of the OPAE releases. To learn more about OPAE, visit https://www.intel.com/content/www/us/en/products/details/fpga/platforms/pac/article.html.

Intel OFS Features	Board Developer	Software Developer	Application Developer
Inherit an ecosystem of Intel-OFS based boards, workloads, and OS distributions	✓	✓	✓
Accelerate workload development with industry-standard Arm AMBA AXI and Avalon® compliant bus interfaces	✓		✓
Accelerate your verification and validation with automated build scripts, a Unified Verification Methodology (UVM) environment and a suite of unit test cases	✓		
Customize your FPGA design (AFU) with modular and composable source code	✓		
Leverage best practices through reference board schematics, schematic layouts, board management and security	✓		✓

For example, the dfl backport driver was packaged with the 1.3.7 release https://github.com/OPAE/opae-sdk/releases/download/1.3.7-4/opae-intel-fpga-driver-2.0.1-8.x86_64.rpm. Like the earlier version, the linux-dfl-backport driver is packaged as an rpm that uses the Dynamic Kernel Module System (DKMS) framework to dynamically rebuild the driver whenever the kernel is updated. For more information on DKMS, see the GitHub project https://github.com/dell/dkms/blob/master/README.md

The linux-dfl-backport branch name is the same as the linux-dfl *-lts branch names. For example, there are a couple for fpga-ofs-dev-5.10-lts maintained by different groups in the community.

Red Hat branches
For RHEL 8 https://github.com/OPAE/linux-dfl-backport/tree/rhel8/fpga-ofs-dev-5.10-lts

For RHEL 9 https://github.com/OPAE/linux-dfl-backport/tree/rhel9/fpga-ofs-dev-5.10-lts

Silicom branch
For n5010, n6010 https://github.com/OPAE/linux-dfl-backport/tree/n5010/fpga-ofs-dev-5.10-lts

The branches are differentiated by the prefixes rhel8/, rhel9/ and n5010/.

The content for the linux-dfl-backport driver is stored in an abbreviated tree in a special base branch. This base branch is created by copying files and the linux long-term branch and cherry picking the linux-dfl changes using the scripts/generate_backport.sh script. Once the group branch is created, the branch maintainer can do normal cherry picking from linux-dfl to keep their branch up to date.

Base branch
https://github.com/OPAE/linux-dfl-backport/tree/base/fpga-ofs-dev-5.10-lts

After the linux-dfl content is imported into the backport driver, there are three more normal tasks to do. First, if there is a new file, the file needs to be added to the top level Makefile.

For example, see commit https://github.com/OPAE/linux-dfl-backport/commit/550bb3fa3d4dc8355b20ce5dbb732d7f3725cc4f

A new file fpga-image-load.c needs to be built. So it is added to the list of 'obj-m''s with

```
obj-m += fpga-image-load.o
```

And then where the source is located

```
fpga-image-load-y := drivers/fpga-image-load.o
```

Then building the driver. The makefile has a help option to see the usage

```
> make help
```

A good target to use is 'reload' as this builds and reinstalls the kernel module.

```
> make reload
```

If all goes well, then use

```
> make rpm
```

to make the rpm.

If this does not work, then the last step of backporting a change is necessary. Although the linux kernel is usually stable, sometimes new changes will depend on features that do not exist or are slightly different from the target kernel version. For example, RHEL 8 is based on the 4.18 kernel whereas the linux-dfl changes are based on the 5.10.y or newer LTS kernel. Steps on backporting are very dependent on the target linux distribution and should be verified against the specific version of the distribution that is needed.

For backporting to mainline kernel ex/ v5.6, the parameters VERSION, PATCHLEVEL, SUBLEVEL are used in the KERNEL_VERSION macro defined in the version header and compared against the LINUX_VERSION_CODE. ex/ For v5.11

```
#define LINUX _ VERSION _ CODE 330496
#define KERNEL _ VERSION(a,b,c) (((a) << 16) +
((b) << 8) + (c))
```

To make a change, add the version header

```
#include <linux/version.h>
…
```

And then isolate your change with an if-def like this check for running on v5.6 or newer

```
#if LINUX _ VERSION _ CODE >= KERNEL _ VERSION
(5, 6, 0)
…
#endif
```

RHEL backporting is complicated because upstream kernel features are backported into RHEL as part of its normal lifecycle. RHEL adds its own version macro to the linux/version.h header.

To make a change, add the version header

```
#include <linux/version.h>
```

ex/ For RHEL 8.3 this is

```
#define LINUX _ VERSION _ CODE 266752
#define KERNEL _ VERSION(a,b,c) (((a) << 16) +
((b) << 8) + (c))
#define RHEL _ MAJOR 8
#define RHEL _ MINOR 3
#define RHEL _ RELEASE _ VERSION(a,b) (((a) << 8)
+ (b))
#define RHEL _ RELEASE _ CODE 2051
#define RHEL _ RELEASE "193.8"
#define KERNEL _ HEADERS _ CHECKSUM
"372410cede0d6cd3b11a85e518dd73a5317f2d58"
```

Because there are no major changes within a specific release of RHEL, it is usually sufficient to check the RHEL_RELEASE_VERSION like this

```
#if RHEL _ RELEASE _ CODE < RHEL _ RELEASE _
VERSION(8,4)
/*  Add the backport code here */
#else
/* Unchanged original code */
#endif
```

When using mainline and RHEL version if-defs, the mainline if-def has precedence and the checkers joined with '&&' for example the above check of running on less than mainline v5.6 or rhel 8.4 would be

```
#if LINUX _ VERSION _ CODE < KERNEL _
VERSION(5.6,0) && RHEL _ RELEASE _ CODE < RHEL _
RELEASE _ VERSION(8,4)
/* Add the backport code here */
#else
/* Unchanged original code */
#endif
```

See how this was done on for RHEL 8.2 to work around the lack of dev_groups in the commit https://github.com/OPAE/linux-dfl-backport/commit/dbb1584db17032ec35148653b7bf62c5f6e89b70

Conclusion

Utilizing this backport driver enables developers to quickly develop an installable driver for new Intel and third-party PACs and SmartNICs based on Intel OFS technology. This backport driver is a community project to enable hardware for immediate development and testing, leveraging Intel OFS capabilities. Before proceeding with using the backport driver, you should consider that this is not a Red Hat supported driver and it should only be used as an interim solution before the driver is accepted in the upstream kernel. This solution should primarily be used to enable your board today with Intel OFS.

Silicom, an Intel and Red Hat partner, have co-developed the community backport driver on their new N5010 and N6010 FPGA SmartNICs enabled with Intel OFS.

"Making the Community Backport Driver enabled Silicom to give customers early access to the new Silicom N5010 and N6010 FPGA SmartNICs using systems with available Red Hat Enterprise Linux 8 releases. The Community Backport Driver provided the required additional handling of the new card in the needed Linux kernels" - Roger Christensen (Silicom, Senior Software Developer)

For More Information

To learn more, visit the following websites:

- Backport driver: https://github.com/OPAE/linux-dfl-backport
- Intel OFS: www.intel.com/OFS
- SIlicom N5010 FPGA SmartNIC: https://www.silicom-usa.com/pr/fpga-based-cards/100-gigabit-fpga-cards/silicom-fpga-smartnic-n5010_series/
- N6010: https://www.silicom-usa.com/pr/fpga-based-cards/100-gigabit-fpga-cards/fpga-smartnic-n6010-intel-based/

WP-01316-1.0