White Paper

Lee Kin Yew
Software Engineer
Intel Corporation

Loading a Splash Screen from *initramfs* into Intel® EMGD DRM as a Binary Blob

January, 2014

330154

Executive Summary

The “splash screen” is an image that appears while a program is loading. It is also an introduction page of the program. One can have any kind of image loaded as a splash screen to embellish any program.

Most programs today have the company logo shown at the start of the program. A splash screen is put in the kernel and loaded at boot time before the program starts. The splash screen image in the kernel must not violate any use of the image license. If the image violates any licensing issue, the splash screen cannot be used in the kernel for any program.

How do we know for certain that the splash screen image violates any licensing issue? How do we use any image as a splash screen and ensure that it will not cause any licensing violation?

A way to do that is to ensure the splash screen is not part of the kernel. But how and where we can place the splash screen image other than in the kernel?

Using the initial RAM file system is the solution to this. The splash screen image will be placed in firmware and loaded at boot time.

Contents

Figures

Business Challenge

In embedded devices, a start-up “splash screen” or “bootup logo” is commonly used in production-ready devices. In embedded IA products that use the Intel® EMGD DRM kernel driver (when Intel® EMGD is built into the kernel), the splash screen has to be built as part of the kernel to ensure that the image loading process succeeds.

However, using this method requires caution as any content that is built into the Linux* kernel will automatically be governed by the General Public License copyright. In short, it will become a “GPL” image. Non-GPL images cannot be bundled into the Linux kernel. This is especially true for images that contain a company’s logo that is trademarked and should not be covered under the GPL.

To avoid legal issues, if you want to use non-GPL splash screen you must ensure that the splash screen is not part of the kernel. For legal reasons, it may be better if non-GPL content remains in a separate file rather than bundled into the kernel.

Intel recommends keeping the splash screen in an *initramfs* rather than in the kernel. Any images, regardless whether they are GPL or non-GPL, can be stored into *initramfs* without triggering any legal issues.

An example will be loading binary blob that contains a splash screen image while we load Intel® EMGD. This will be more useful for the user who builds Intel® EMGD as a built-in kernel. To get the binary blob loaded when the graphics driver is loaded, you must store the splash screen inside a binary blob that will be built into the kernel. With this method, when Intel® EMGD loads, the driver will look for a binary blob that contains the splash screen and then load it. This will definitely cause legal issues if the splash screen is a non-GPL image.

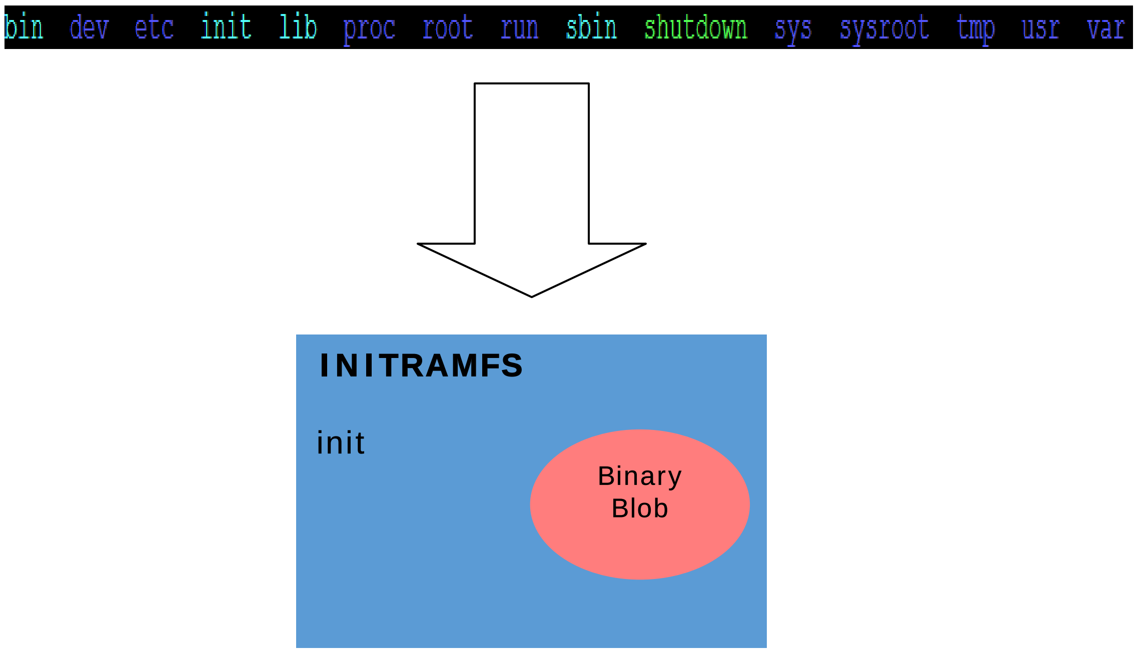
This white paper is the solution for using non-GPL as splash screen in a binary blob when the graphics driver loads and uses the binary blob. The splash screen will be placed into the binary blob and stored into an *initramfs*.

Solution

A solution to a non-GPL or trademarked image being loaded as a splash screen with Intel® EMGD built into the kernel driver is to package the splash screen binary blob into the initial RAM file system, *initramfs*. An *initramfs* is based on *tmpfs*, which does not use a separate block device. It contains the tools and scripts needed to mount the file systems before the init binary from the real root file system is executed. This approach does not violate the use of non-GPL image as it is not part of the Linux kernel.

The content of *initramfs* is made by creating a cpio archive. All files, tools, libraries, and configuration settings are put into the cpio archive.

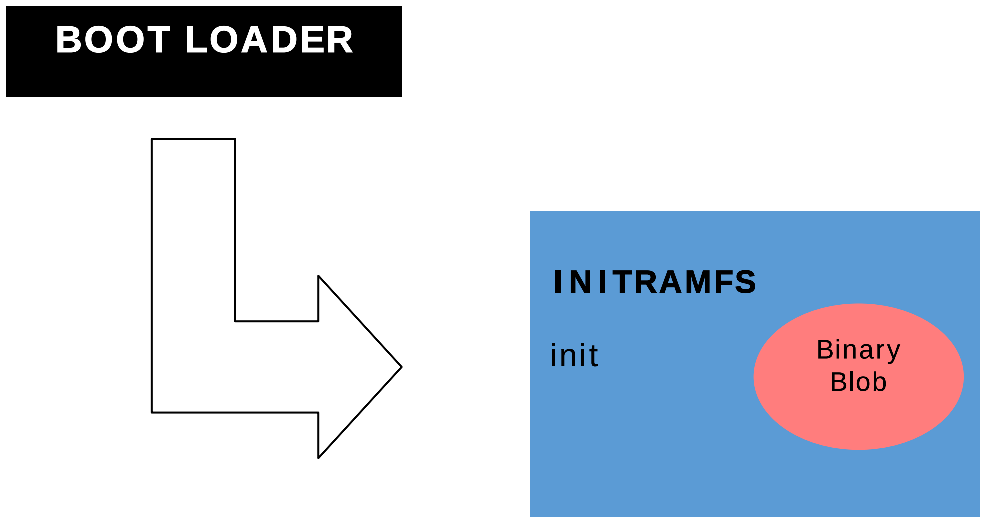
Figure 1. Content of initramfs packed into cpio archive



```
bin  dev  etc  init  lib  proc  root  run  sbin  shutdown  sys  sysroot  tmp  usr  var
```

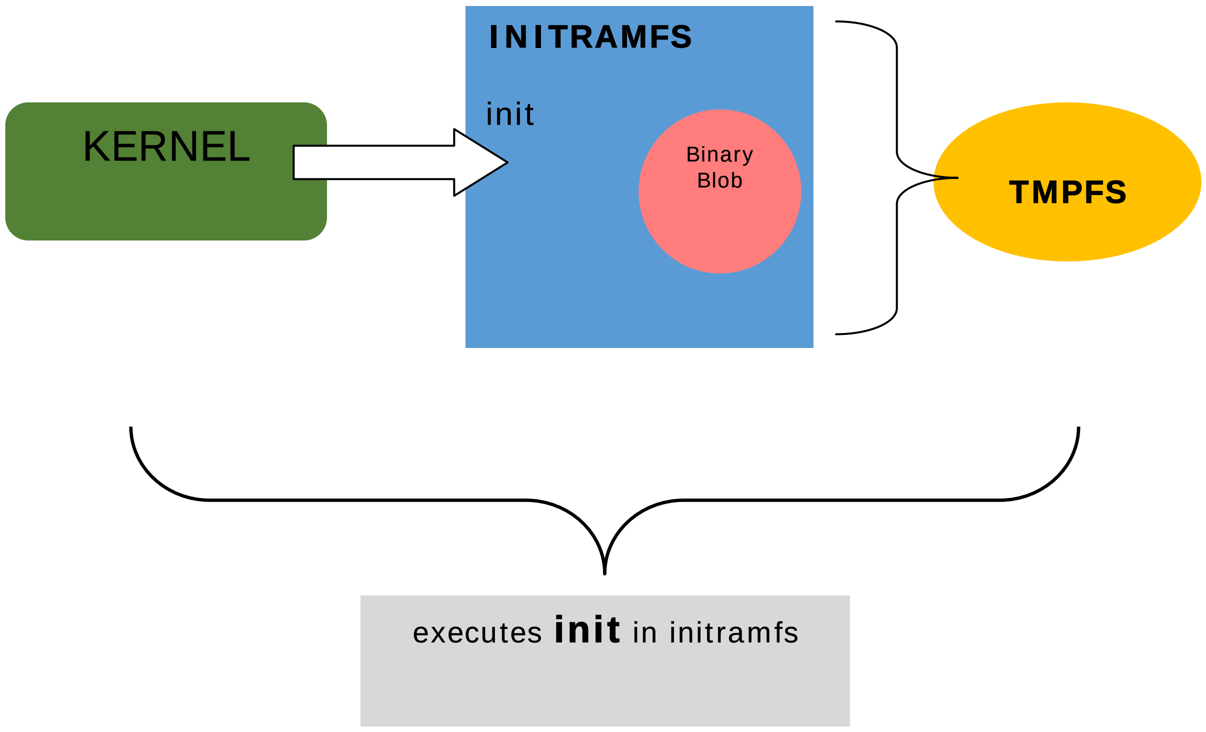
This archive is then compressed with gzip and stored in the Linux kernel. The boot loader will then offer it to the Linux kernel at boot time and the kernel knows an *initramfs* is presented.

Figure 2. Boot loader loads *initramfs* at boot time



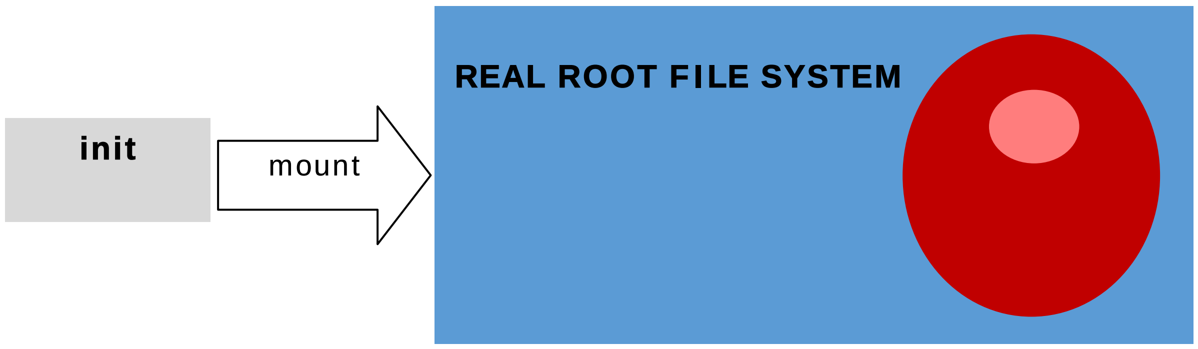
Once detected, the Linux kernel will create a *tmpfs* file system, extract the contents of the archive from it, and then launch the init script located in the root of the *tmpfs* file system.

Figure 3. Execute the init srcipt from the tmpfs



The script will then mount the real root file system as well as other vital file systems such as */usr* and */var*.

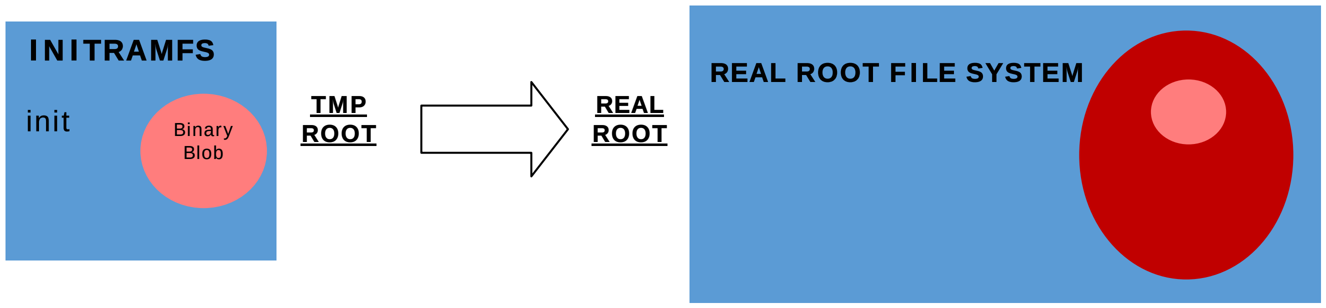
Figure 4. Mount the real root file system



The kernel "rootfs" becomes the *tmpfs* that the *initramfs* is unpacked into if *initramfs* is used.

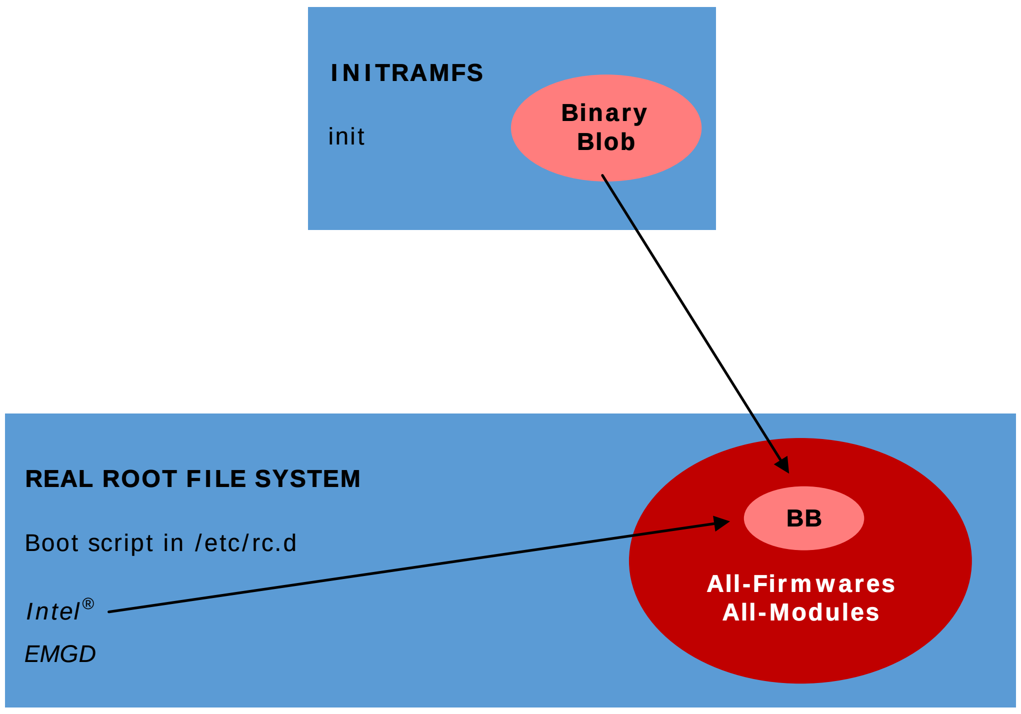
After the root file system and the other vital file systems are mounted, the init script from the *initramfs* switches the temporary root from the *initramfs* to the real root file system, and finally calls the */sbin/init* on that system to continue the boot process.

Figure 5. Switch from tmp root to the real root



During the boot process after the root is switched from *initramfs* to the real root file system, Intel® EMGD will be loaded as it is built into the kernel. Intel® EMGD will look for the binary blob inside */lib*.

Figure 6. Looking for binary blob when emgd.ko loads



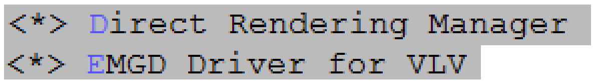
Solution Methods

1. Compile Intel® EMGD as kernel built-in driver:

   a. Make Intel® EMGD as built-in driver in kernel source compilation:

```
cd <kernel_src>
make menuconfig
```

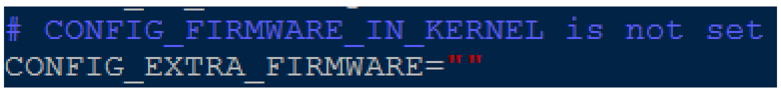
   In Device Drivers -> Graphics support, put an asterisk (<*>) instead of <M> for DRM and EMGD.



```
<*> Direct Rendering Manager
<*> EMGD Driver for VLV
```

   Save and exit.

   b. Ensure there is no binary blob configured as a kernel built-in in the kernel configuration:



```
# CONFIG_FIRMWARE_IN_KERNEL is not set
CONFIG_EXTRA_FIRMWARE=""
```

   c. Compile kernel source code:

```
make
```

   d. After the compilation is completed, copy the bzImage (kernel image) to */boot*:

```
cp <kernel_src>/arch/x86/boot/bzImage /boot/vmlinuz-version-emgd- built-in
```

2. Create *initrmafs* with the binary blob inside:
   a. You can either build a new *initramfs* or modify the existing one.
   b. To create a new *initramfs*, you need an application named "dracut" or any *initramfs* script.
   c. Create a new *initramfs*:
      <u>Fedora</u> – Use dracut to create *initramfs*.

```
dracut initramfs-XXX.img kernel_name
```

      <u>Tizen</u> – Create a script to make *initramfs* if dracut is not available in Tizen.

```
mkinitramfs kernel_name
```

   d. Create a folder to store the extracted contents of *initramfs*:

```
mkfir /root/initramfs_folder
cd /root/initramfs_folder
```

   e. Extract the contents of *initramfs* into the folder.

```
gunzip < /boot/initramfs-XXX.img | cpio -i
```

   f. You will have the *initramfs* contents in the *initramfs* folder now. The next step is to store the binary blob, *emgd.bin* into the *initramfs*:

```
cd /root/initramfs_folder/lib
mkdir firmware
cd firmware
cp <path_to_local_binary_blob>/emgd.bin .
cd ../../../
```

g. Pack all the contents in the *initramfs* folder back to an archive file.

```
find | cpio -H newc -o | gzip -9 > /boot/initramfs-XXX-with-binaryblob.img
```

h. Now there is a new *initramfs* with the binary blob inside, which is placed in */boot*.

3. Configure the boot menu:

a. Edit the boot loader menu so it loads the requested kernel and *initramfs*. Please ensure that:

1. Kernel loads the EMGD built-in image:

```
linux /boot/vmlinuz-version-emgd-built-in
```

2. emgd.firmware as a linux/ parameter points to the correct name of the binary blob:

```
emgd.firmware="emgd.bin"
```

3. initrd points to the correct *initramfs*:

```
initrd /boot/initramfs-XXX-with-binaryblob.img
```

Example extracted from grub.conf

```
 linux   /boot/vmlinuz-version-emgd-built-in root=UUID=686ab39c-a960-47b9-9982-6564dd681465
emgd.firmware="emgd.bin" ro rd.md=0 rd.lvm=0 rd.dm=0  rd.luks=0 vconsole.keymap=us LANG=en_
US.UTF-8
echo 'Loading initial ramdisk ...'
initrd  /boot/initramfs-XXX-with-binaryblob.img
```

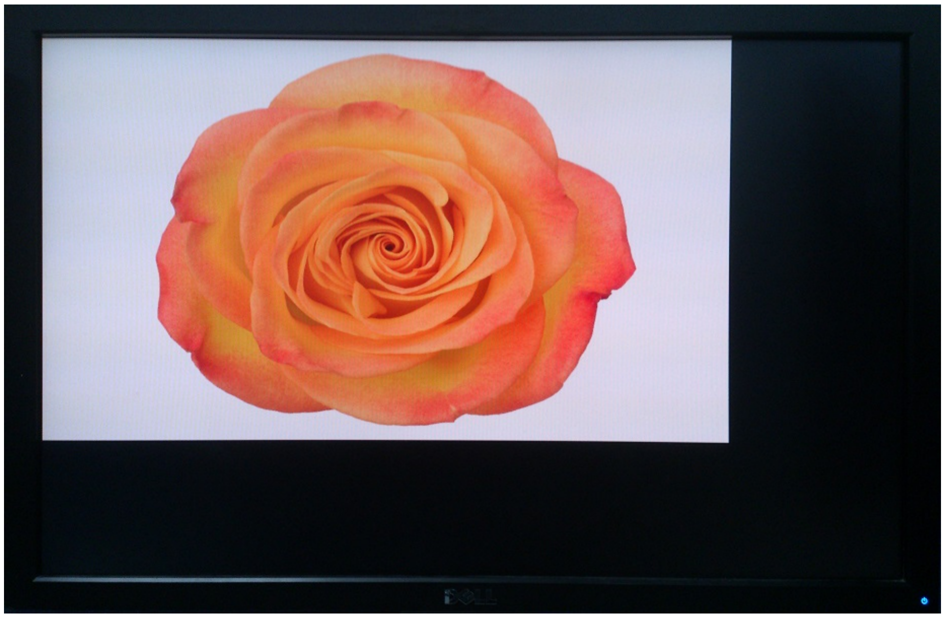
Example extracted from extlinux.conf

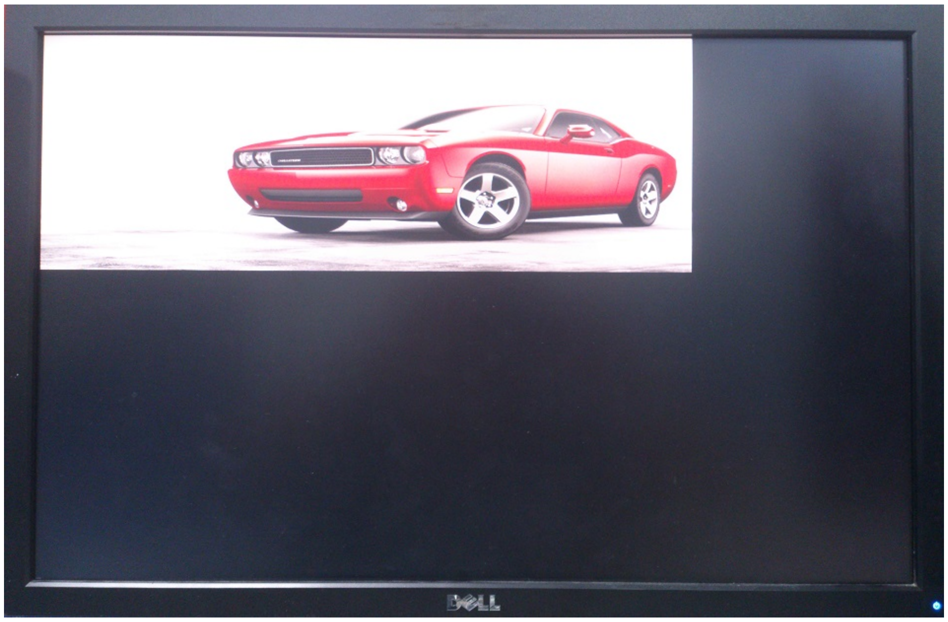
```
linux /vmlinuz-version-emgd-built-in
append ro root=/dev/sda3 emgd.firmware="emgd.bin" rootwait rootfstype=ext4 quiet
initrd /initramfs-XXX-with-binaryblob.img
```

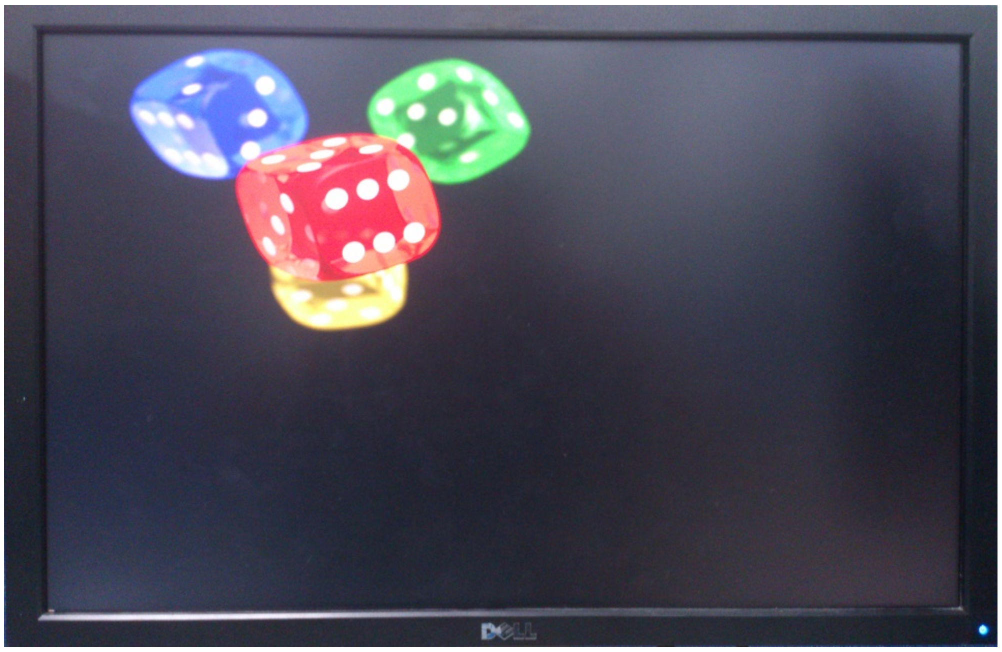
Splash Screen of GPL and non-GPL Images

Results of images when the binary blob is loaded during the Intel® EMGD loading time.

GPL Images







NON-GPL Images

The Intel logo is trademarked and hence is not a GPL logo, which cannot be bundled into the kernel. This Intel logo is placed in an `initramfs` as a binary blob.

Building Environment

Fedora

Dracut is pre-installed in most of the Fedora distro versions.

Link to Dracut
https://dracut.wiki.kernel.org/

Tizen

Dracut should not be pre-installed in Tizen. You can install it via zypper. An alternative way is to create a script to create the *initramfs*. You can refer to the recommended link below to learn how to create a script to make an *initrmafs*.

Link to create an initramfs script
http://www.linuxfromscratch.org/blfs/view/svn/postlfs/initramfs.html

Time Taken to Load Splash Screen

There is no significant time differences between loading a splash screen from `initramfs` and the kernel. Time is taken from loading Intel® EMGD until the splash screen is displayed on the screen. Time measured loading from both `initramfs` or kernel is ~0.97s.

Conclusion

The solution proposed in this paper is a way for the customer to use any non-GPL image as a splash screen when Intel® EMGD is a kernel built-in driver. This will avoid the violation of using non-GPL content in the Linux kernel.

Authors

Lee Kin Yew is a Software Engineer at Intel Corporation.

Acronyms

DRM	Direct Rendering Manager
EMGD	Intel® Embedded and Media Graphics Driver
GPL	General Public License