How to create a partitioned image with the custom Wic plugin?

Tips and tricks based on the bootimg-grub-tb plugin development

Norbert Kamiński,
3mdeb Embedded Systems Consulting

Yocto Project *Virtual* Summit Europe, October 29-30, 2020

Agenda

- ***$ whoami***
- **Wic – OpenEmbedded Image Creator**
  - Preparing layer
  - WKS files
- **Wic Plug-in Interface**
  - Overall information
  - PluginSource Methods
- **Wic Plug-in development**
  - bootimg-grub-tb - custom Wic Plug-in

$ whoami

Norbert Kamiński

Embedded Systems Engineer at

3mdeb Embedded Systems Consulting

- **Open-source contributor**
  - meta-pcengines
  - meta-trenchboot
  - qubes-fwupd
- **Scope of interests**
  - embedded Linux
  - virtualization and containerization
  - bootloaders

Wic – OpenEmbedded Image Creator

What is the Wic?

- Wic stands for OpenEmbedded Image Creator
- It is used to a create partitioned image
- Wic is loosely based on the Meego Image Creator framework (mic)
- It is using build artifacts instead of installing packages and configurations

Prepare your layer

- Go to your meta layer
- Add wic to the IMAGE_FSTYPE variable in your local configuration

```
IMAGE_FSTYPES += "wic"
```

- Use the existing wic kickstart file or create specific one for your purposes

Default partition layouts

- At the start source poky/oe-init-build-env
- List the available wic kickstart configurations

```
$ wic list images
  mpc8315e-rdb                          Create SD card image for MPC8315E-RDB
  edgerouter                            Create SD card image for Edgerouter
  beaglebone-yocto                      Create SD card image for Beaglebone
  genericx86                            Create an EFI disk image for genericx86*
[...]
  qemuriscv                             Create qcow2 image for RISC-V QEMU machines
```

- Choose the WKS configuration and add it to the local.conf file

```
WKS_FILE = "genericx86.wks"
```

Create your custom partition layout

- Go to your layer and create the `wic` directory
- Create Wic kickstart file and set the description for your partition layout

```
# short-description: Create a partitioned image for the TrenchBoot
# long-description: Create a partitioned image for the TrenchBoot
```

- Specify your custom partition layout

```
part /boot  --source bootimg-grub-tb  --ondisk sda --label msdos --active --align 1024
part /      --source rootfs --ondisk sda --fstype=ext4 --label rootfsA1 --align 4096 --fixed-size 2048
part        --source rootfs --ondisk sda --fstype=ext4 --label rootfsA2 --align 4096 --fixed-size 2048
```

Create your custom partition layout

```
part /boot  --source bootimg-grub-tb  --ondisk sda --label msdos --active --align 1024
part /      --source rootfs --ondisk sda --fstype=ext4 --label rootfsA1 --align 4096 --fixed-size 2048
part        --source rootfs --ondisk sda --fstype=ext4 --label rootfsA2 --align 4096 --fixed-size 2048
```

- `part` **command creates partition and takes the mount point as the input e.g.** `/boot`
- `--source` **parameter specifies the Wic plug-in**
- `--ondisk` **parameter force creating the partition on a particular disk**
- `--fstype` **sets the file system for the partition**

Create your custom partition layout

```
part /boot  --source bootimg-grub-tb  --ondisk sda --label msdos --active --align 1024
part /      --source rootfs --ondisk sda --fstype=ext4 --label rootfsA1 --align 4096 --fixed-size 2048
part        --source rootfs --ondisk sda --fstype=ext4 --label rootfsA2 --align 4096 --fixed-size 2048
```

- `--label` specifes the the label that is given for the filesystem
- `--active` sets the partition as bootable
- `--align` specifies maximum size of boundaries between the partitions (in KB)
- `--fixed-size` sets exact size of the partition (in MB)

Create your custom partition layout

- **Specify the bootloader options**

```
bootloader  --ptable msdos --timeout=5 --append="rootfstype=ext4 console=ttyS0,115200 earlyprintk=serial,ttyS0,115200"
```

- **If you list the available partition configurations, you will see the new custom configuration**

```
$ wic list images
  trenchboot-image                  Create a partitioned image for the TrenchBoot
  mpc8315e-rdb                      Create SD card image for MPC8315E-RDB
  edgerouter                        Create SD card image for Edgerouter
  beaglebone-yocto                  Create SD card image for Beaglebone
  genericx86                        Create an EFI disk image for genericx86*
[...]
  qemuriscv                         Create qcow2 image for RISC-V QEMU machines
```

Development tips

- **You don’t need to flash the device to check, if the partition layout is correct. Use the loop device to this purpose:**

```
$ sudo losetup -P -f --show build/tmp/deploy/images/pcengines-apu2/xen-tb-dom0-image-pcengines-apu2.wic
/dev/loop20
$ sudo fdisk -l /dev/loop20
[...]
Disk identifier: 0x34c2f22d

Device         Boot   Start     End Sectors  Size Id Type
/dev/loop20p1 *       2048   89687   87640 42,8M  c W95 FAT32 (LBA)
/dev/loop20p2        90112 4284415 4194304    2G 83 Linux
/dev/loop20p3      4284416 8478719 4194304    2G 83 Linux
```

Development tips

- To speed up the flashing process create the bmap artifact. Add wic.bmap and wic.gz to the IMAGE_FSTYPE

```
IMAGE_FSTYPES += "wic wic.gz wic.bmap"
```

- Use bmap-tool to copy the image on the drive

```
# bmaptool copy --bmap xen-tb-dom0-image-efi-genericx86-64.wic.bmap xen-tb-dom0-image-efi-genericx86-64.wic.gz \
/dev/sdd
bmaptool: info: block map format version 2.0
bmaptool: info: 540672 blocks of size 4096 (2.1 GiB), mapped 139322 blocks (544.2 MiB or 25.8%)
bmaptool: info: copying image 'xen-tb-dom0-image-efi-genericx86-64.wic.gz' to block device '/dev/sdd'
bmaptool: info: using bmap file 'xen-tb-dom0-image-efi-genericx86-64.wic.bmap'
bmaptool: info: 100% copied
bmaptool: info: synchronizing '/dev/sdd'
bmaptool: info: copying time: 1m 37.3s, copying speed 5.6 MiB/sec
```

Wic Plug-in Interface

Wic Plug-in Interface

- **Wic plug-in interface provides the mechanism to customize the image generation process**
- **`--source` variable in the Wic kickstart file specifies the the source plugin which is used to generate the partition image**
- **The source plugins are subclasses based on `SourcePlugin` class, which is defined in `poky/scripts/lib/wic/pluginbase.py`**

Wic Plug-in Interface

- **Wic plug-in sources could be defined in the external layers**
- **Custom plugins must be placed in `scripts/lib/wic/plugins/source/` within external layer**
- **Each Wic plug-in has particular name variable that corresponds to the `--source` parameter**

```
class BootimgGrubTbPlugin(SourcePlugin):
    """
    Creates TrenchBoot boot partition for PC BIOS platforms
    """
    name = 'bootimg-grub-tb'
```

The methods of the SourcePlugin class

The `SourcePlugin` class provides the following methods to the source file:

- **`do_configure_partition()` - method that creates custom configuration files for a partition (e. g. custom grub.cfg file)**
- **`do_stage_preparation()` - method allows stage the partition files in customized way. Typically, this method is empty.**

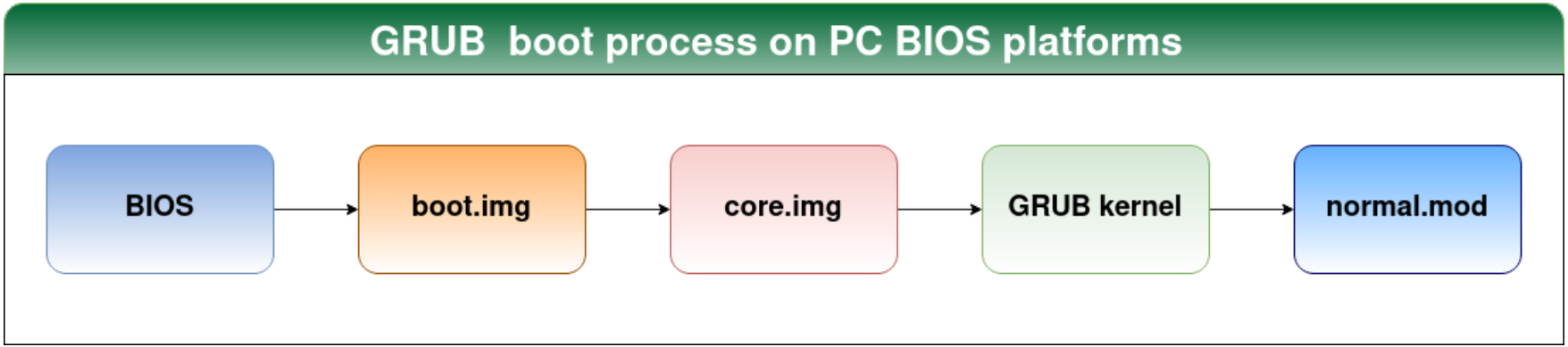
The methods of the SourcePlugin class

- `do_prepare_partition()` - method does the content population for a partition, it prepares the final partition to be incorporated into image
- `do_install_disk()` - method finalize the disk image creation, e. g. it writes the MBR (Master Boot Record)
- `do_post_partition()` - method allows to execute the post operations after the partition is created, it could be used e. g. for security signing

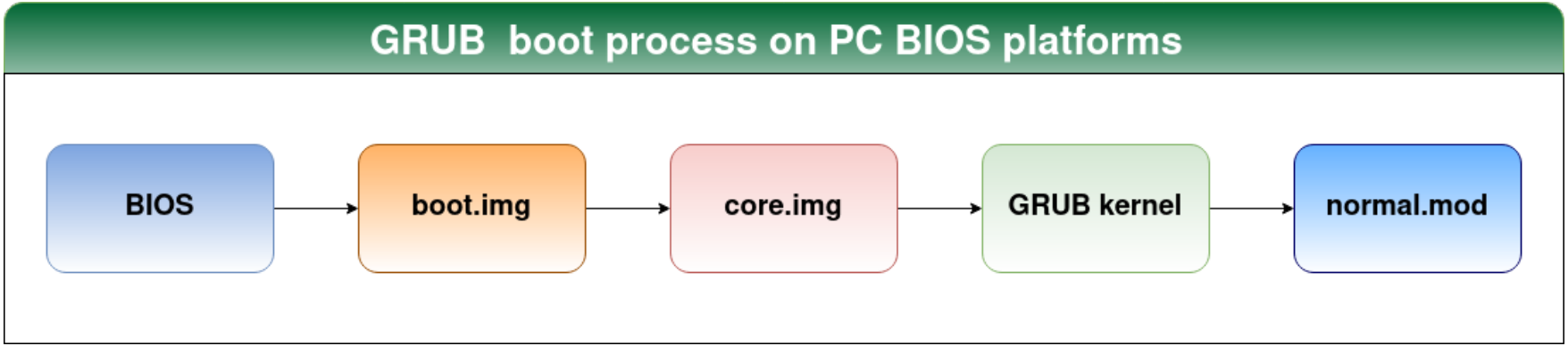
Wic Plug-in development

Custom boot partition for the TrenchBoot

- **TrenchBoot is a framework that allows individuals and projects to build security engines to perform launch integrity actions for their systems**
- **The meta-trenchboot layer creates ready to use Yocto builds, that provides the D-RTM for the UEFI and PC BIOS platforms**
- **bootimg-grub-tb plug-in creates the custom boot partition for the TrechBoot purposes**

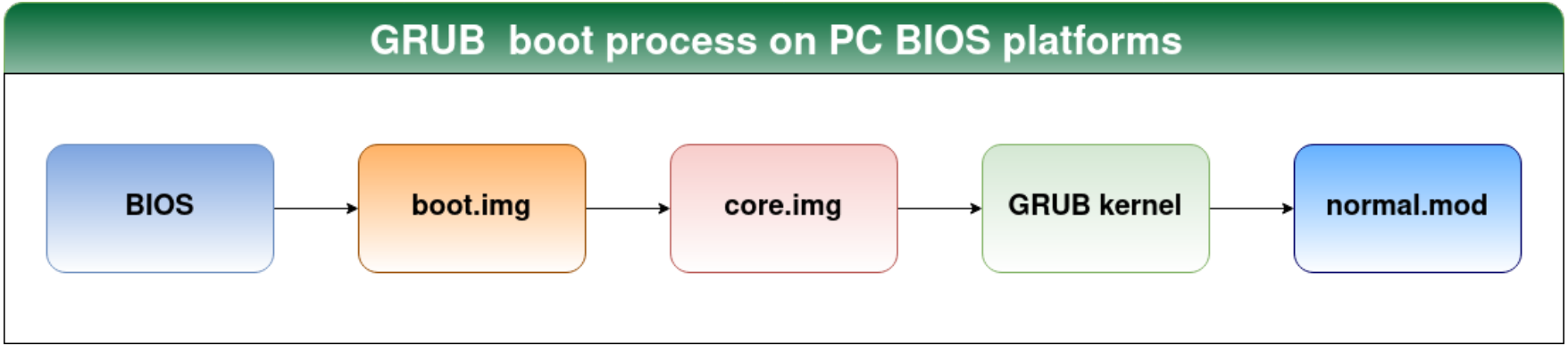


- **The BIOS selects a hard drive to boot from and loads the boot image**
- **The boot image is written to the first 512-byte sector of the partition. It contains logical block addressing (LBA) of the first sector of core image and its task is to load that sector into memory and transfer the control to core image**



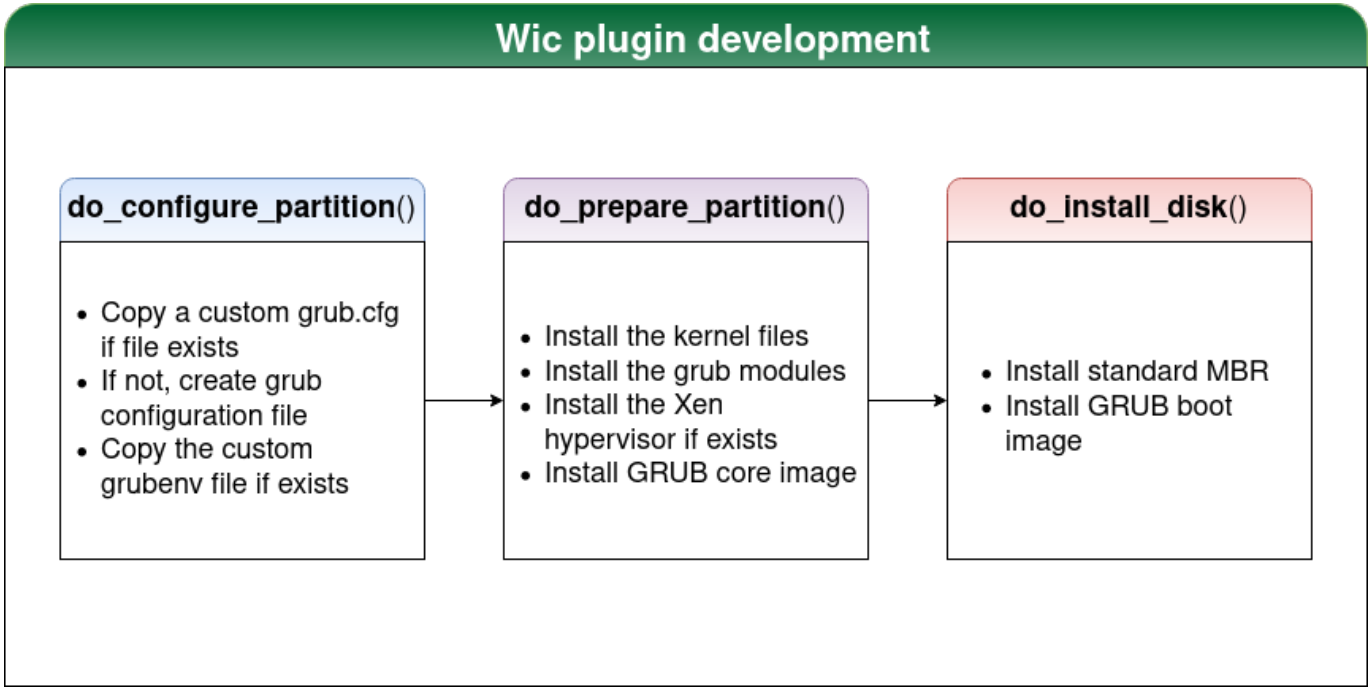
- **The core image contains the loader and addresses of the blocks that allows loading core image to the memory. Once it is done the GRUB kernel takes over initialization.**

- **At first GRUB is trying to load the `normal` module. It attempts to locate the `normal.mod file` that should be installed in the `(hd0,msdos1)/boot/grub/i386-pc/` directory.**



- **Normal module loads another couple of modules. Once the normal is loaded properly, it runs the normal command that displays the GRUB menu with the grub.cfg configuration file.**

bootimg-grub-tb – Wic plug-in development



The plug-in source: https://github.com/3mdeb/meta-trenchboot/blob/master/scripts/lib/wic/plugins/source/bootimg-grub-tb.py

Development Challenges

- `grub-bios-setup` **allows to install grub boot image to the physical device. When creating the bootable partition in a file, it's not possible to guess the root partition. To workaround this problem, it's necessary to create new parameter (`-r`) that would specify the root partition**

```
grub_dir = os.path.join(workdir, "hdd/boot/grub/i386-pc")
cmd_bios_setup = (
    f'grub-bios-setup -v --device-map={device_map_path}'
    f' -r "hd0,msdos1" -d {grub_dir} {full_path}'
)
exec_cmd(cmd_bios_setup, native_sysroot)
```

The patch source code: https://github.com/3mdeb/meta-trenchboot/blob/master/dynamic-layers/openembedded-layer/recipes-bsp/grub/grub-tb/0001-add-root-flag-to-grub-bios-setup.patch

Development Tips

- To obtain Bitbake variables use `get_bitbake_var`

```
deploy_dir = get_bitbake_var("DEPLOY_DIR_IMAGE")
```

- The second way of passing variables to the plug-in is `--sourceparams` parameter; it is defined in the Wic kickstarter file

```
part /boot  --source bootimg-grub-tb --sourceparams="initrd=initrd.cpio" \
 --ondisk sda --label msdos --active --align 1024
```

To obtain the parameter in the plugin use `source_params.get`

```
initrd = source_params.get('initrd')
```

Development Tips

During the plug-in development you may need to use the parameters of SourcePlugin methods:

- `cr_workdir` – **work directory of the Wic, it contains created artifacts**
- `hdddir` – **directory which is used to populate a partition**
- `kernel_dir` – **kernel directory typically it's equal to** `DEPLOY_DIR_IMAGE`

Thanks for your time





yocto
PROJECT

THE
LINUX
FOUNDATION