LEDGE reference platfrom documentation

Release unknown-rev

Linaro Limited and Contributors

Mar 11, 2020

CONTENTS

Table 1: Revision History

Date	Issue	Changes
17 Febrary 2020	0.1	• Initial version

CHAPTER

ONE

ABOUT THIS DOCUMENT

1.1 Introduction

The LEDGE documentation describes instructions to build, install and use various features included in LEDGE linux reference platform.

Comments or change requests can be sent to team-ledge@linaro.org.

1.2 Guiding Principles

This documentation describes how to build fully open source version of LEDGE reference platform and run it.

1.3 Scope

LEDGE reference platform is intended for IoT and EDGE devices. So that support of high level of security takes major place. Document only describes high level of features usage and implementation details. This document does not provide full technical documentation about specific features.

1.4 Cross References

This document cross-references sources that are listed in the References section by using the section sign §.

Examples:

UEFI § 6.1 - Reference to the UEFI specification [UEFI] section 6.1

1.5 Terms and abbreviations

This document uses the following terms and abbreviations.

A64 The 64-bit Arm instruction set used in AArch64 state. All A64 instructions are 32 bits.

AArch32 Arm 32-bit architectures. AArch32 is a roll up term referring to all 32-bit versions of the Arm architecture starting at ARMv4.

AArch64 state The Arm 64-bit Execution state that uses 64-bit general purpose registers, and a 64-bit program counter (PC), Stack Pointer (SP), and exception link registers (ELR).

AArch64 Execution state provides a single instruction set, A64.

EFI Loaded Image An executable image to be run under the UEFI environment, and which uses boot time services.

EL0 The lowest Exception level on AArch64. The Exception level that is used to execute user applications, in Non-secure state.

EL1 Privileged Exception level on AArch64. The Exception level that is used to execute Operating Systems, in Non-secure state.

EL2 Hypervisor Exception level on AArch64. The Exception level that is used to execute hypervisor code. EL2 is always in Non-secure state.

EL3 Secure Monitor Exception level on AArch64. The Exception level that is used to execute Secure Monitor code, which handles the transitions between Non-secure and Secure states. EL3 is always in Secure state.

Logical Unit (LU) A logical unit (LU) is an externally addressable, independent entity within a device. In the context of storage, a single device may use logical units to provide multiple independent storage areas.

OEM Original Equipment Manufacturer. In this document, the final device manufacturer.

SiP Silicon Partner. In this document, the silicon manufacturer.

UEFI Unified Extensible Firmware Interface.

UEFI Boot Services Functionality that is provided to UEFI Loaded Images during the UEFI boot process.

UEFI Runtime Services Functionality that is provided to an Operating System after the ExitBootServices() call.

CHAPTER

TWO

LEDGE OVERVIEW

2.1 General

LEDGE images are related to IoT and EDGE devices. It has advanced security features supported:

- Secure UEFI boot
- OP-TEE (Open Portable Trusted Execution Environment)
- ARM trusted Firmware (AT-F)
- Teanocore edk2 firmware or Uboot with UEFI mode support
- fTMP (Firmware TPM driver with backend to OP-TEE)
- Kernel image sign with certificate
- Kernel modules sign
- IMA/EVM for integrity user applications
- Selinux
- Containered isolation (docker)
- Advanced system update

The LEDGE image consist of WIC image and firmware to boot this image on specific board or virtual machine.

2.2 WIC image

LEDGE WIC image for consists of 2 partiotions - ESP partition and linux rootfs partition. Disk image may have gpt lable type or may not, depends on varios hardware requirements.

```
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 5B707C60-D281-441C-A31C-73849DD89C49

Device          Start     End Sectors   Size Type
/dev/loop6p1       40  123319  123280  60.2M Microsoft basic data
/dev/loop6p2   123320 1861731 1738412 848.9M Linux filesystem
```

2.3 Guiding Principles

This documentation describes how to build fully open source version of LEDGE reference platform and run it.

2.4 Scope

LEDGE reference platform is intended for IoT and EDGE devices. So that support of high level of security takes major place. Document only describes high level of features usage and implementation details. This document does not provide full technical documentation about specific features.

2.5 Cross References

This document cross-references sources that are listed in the References section by using the section sign §.

Examples:

UEFI § 6.1 - Reference to the UEFI specification [UEFI] section 6.1

2.6 Terms and abbreviations

This document uses the following terms and abbreviations.

2.6.1 ESP partition

ESP partition is about 60 Megabytes vfat partition wit the following structure:

```
├── dtb
├── EFI
│   └── BOOT
│       └── bootarm.efi
└── ledge-initramfs.rootfs.cpio.gz
```

Where:

- dtb - directory which contains all dtbs for all supported devices.
- bootarm.efi - linux kernel compiled as UEFI stab and which boots directly from firmware. (bootx64.efi for x86, bootaarch.efi)
- ledge-initramfs.rootfs.cpio.gz - initramfs is used to do initial initialization and find and mount rootfs.

CHAPTER

THREE

OPENEMBEDDED

This chapter describes specific OpenEmbedded LEDGE build and run.

3.1 Supported platfroms

- armv7/ledge-multi-armv7 (qemu, ti-am572x, stm32mp157c-dk2);
- armv8/ledge-multi-armv8 (qemu, synquacer)
- x86-64 (qemu)

3.2 Build steps

3.2.1 Download sources:

```
repo init --no-clone-bundle --depth=1 --no-tags -u https://github.com/Linaro/ledge-
↪oe-manifest.git -b master
repo sync
```

3.2.2 Setup environment and run build:

3.2.3 armv7 family:

```
MACHINE=ledge-multi-armv7 DISTRO=rpb source ./setup-environment build-rpb
bitbake mc:qemuarm:ledge-iot mc:qemuarm:ledge-gateway ${FIRMWARE}
```

Image files will apper under: armhf-glibc/deploy/images directory.

Generated output will be:

```
├── ledge-qemuarm
│   ├── arm-trusted-firmware
│   │   ├── bl1.bin
│   │   ├── bl1.elf
│   │   ├── bl2.bin
│   │   └── bl2.elf
│   ├── bl1.bin -> arm-trusted-firmware/bl1.bin
│   ├── bl2.bin -> arm-trusted-firmware/bl2.bin
│   ├── bl32.bin -> optee/tee-header_v2.bin
│   ├── bl32_extra1.bin -> optee/tee-pager_v2.bin
│   ├── bl32_extra2.bin -> optee/tee-pageable_v2.bin
│   ├── bl33.bin -> u-boot-ledge-qemuarm.bin
│   ├── dtb
```

(continues on next page)

(continued from previous page)

```
│   ├── kernel-devicetrees.tgz
│   ├── ledge-gateway.env
│   ├── ledge-gateway-ledge-kernel-uefi.wks
│   ├── ledge-gateway-ledge-qemuarm-20200218104425.bootfs.vfat
│   ├── ledge-gateway-ledge-qemuarm-20200218104425.bootfs.vfat.gz
│   ├── ledge-gateway-ledge-qemuarm-20200218104425.qemuboot.conf
│   ├── ledge-gateway-ledge-qemuarm-20200218104425.rootfs.manifest
│   ├── ledge-gateway-ledge-qemuarm-20200218104425.rootfs.wic
│   ├── ledge-gateway-ledge-qemuarm-20200218104425.testdata.json
│   ├── ledge-gateway-ledge-qemuarm.bootfs.vfat -> ledge-gateway-ledge-qemuarm-
↪20200218104425.bootfs.vfat
│   ├── ledge-gateway-ledge-qemuarm.bootfs.vfat.gz
│   ├── ledge-gateway-ledge-qemuarm.manifest -> ledge-gateway-ledge-qemuarm-
↪20200218104425.rootfs.manifest
│   ├── ledge-gateway-ledge-qemuarm.qemuboot.conf -> ledge-gateway-ledge-qemuarm-
↪20200218104425.qemuboot.conf
│   ├── ledge-gateway-ledge-qemuarm.testdata.json -> ledge-gateway-ledge-qemuarm-
↪20200218104425.testdata.json
│   ├── ledge-gateway-ledge-qemuarm.wic -> ledge-gateway-ledge-qemuarm-
↪20200218104425.rootfs.wic
│   ├── ledge-initramfs-ledge-qemuarm.cpio.gz -> ledge-initramfs.rootfs.cpio.gz
│   ├── ledge-initramfs-ledge-qemuarm.manifest -> ledge-initramfs.rootfs.manifest
│   ├── ledge-initramfs-ledge-qemuarm.qemuboot.conf -> ledge-initramfs.qemuboot.
↪conf
│   ├── ledge-initramfs-ledge-qemuarm.testdata.json -> ledge-initramfs.testdata.
↪json
│   ├── ledge-initramfs.qemuboot.conf
│   ├── ledge-initramfs.rootfs.cpio.gz
│   ├── ledge-initramfs.rootfs.manifest
│   ├── ledge-initramfs.testdata.json
│   ├── ledge-iot.env
│   ├── ledge-iot-ledge-kernel-uefi.wks
│   ├── ledge-iot-ledge-qemuarm-20200218104425.bootfs.vfat
│   ├── ledge-iot-ledge-qemuarm-20200218104425.bootfs.vfat.gz
│   ├── ledge-iot-ledge-qemuarm-20200218104425.qemuboot.conf
│   ├── ledge-iot-ledge-qemuarm-20200218104425.rootfs.manifest
│   ├── ledge-iot-ledge-qemuarm-20200218104425.rootfs.wic
│   ├── ledge-iot-ledge-qemuarm-20200218104425.testdata.json
│   ├── ledge-iot-ledge-qemuarm.bootfs.vfat -> ledge-iot-ledge-qemuarm-
↪20200218104425.bootfs.vfat
│   ├── ledge-iot-ledge-qemuarm.bootfs.vfat.gz
│   ├── ledge-iot-ledge-qemuarm.manifest -> ledge-iot-ledge-qemuarm-20200218104425.
↪rootfs.manifest
│   ├── ledge-iot-ledge-qemuarm.qemuboot.conf -> ledge-iot-ledge-qemuarm-
↪20200218104425.qemuboot.conf
│   ├── ledge-iot-ledge-qemuarm.testdata.json -> ledge-iot-ledge-qemuarm-
↪20200218104425.testdata.json
│   ├── ledge-iot-ledge-qemuarm.wic -> ledge-iot-ledge-qemuarm-20200218104425.
↪rootfs.wic
│   ├── ledge-kernel-uefi-certs.ext4.img
│   ├── ledge-qemuarm.dtb
│   ├── modules-ledge-qemuarm.tgz -> modules--mainline-5.3-r0-ledge-qemuarm-
↪20200218104425.tgz
│   ├── modules--mainline-5.3-r0-ledge-qemuarm-20200218104425.tgz
│   ├── modules-stripped-ledge-qemuarm-for-debian.tgz
│   ├── modules-stripped-ledge-qemuarm.tgz -> modules-stripped--mainline-5.3-r0-
↪ledge-qemuarm-20200218104425.tgz
│   ├── modules-stripped--mainline-5.3-r0-ledge-qemuarm-20200218104425.tgz
│   ├── optee
│   │   ├── tee.bin
│   │   ├── tee-header_v2.bin
```

(continues on next page)

(continued from previous page)

```
│       │   ├── tee-pageable.bin
│       │   ├── tee-pageable_v2.bin
│       │   ├── tee-pager.bin
│       │   └── tee-pager_v2.bin
│       ├── u-boot-basic-1.0-r0.bin
│       ├── u-boot.bin -> u-boot-basic-1.0-r0.bin
│       ├── u-boot.bin-basic -> u-boot-basic-1.0-r0.bin
│       ├── u-boot-ledge-qemuarm.bin -> u-boot-basic-1.0-r0.bin
│       ├── u-boot-ledge-qemuarm.bin-basic -> u-boot-basic-1.0-r0.bin
│       ├── zImage -> zImage--mainline-5.3-r0-ledge-qemuarm-20200218104425.bin
│       ├── zImage-for-debian
│       ├── zImage-ledge-qemuarm.bin -> zImage--mainline-5.3-r0-ledge-qemuarm-
↪20200218104425.bin
│       └── zImage--mainline-5.3-r0-ledge-qemuarm-20200218104425.bin
├── ledge-stm32mp157c-dk2
│   ├── arm-trusted-firmware
│   │   ├── bl2.bin
│   │   ├── bl2.elf
│   │   └── tf-a-stm32mp157c-dk2.stm32
│   ├── optee
│   │   ├── tee.bin
│   │   ├── tee-header_v2.bin
│   │   ├── tee-header_v2.stm32
│   │   ├── tee-pageable.bin
│   │   ├── tee-pageable_v2.bin
│   │   ├── tee-pageable_v2.stm32
│   │   ├── tee-pager.bin
│   │   ├── tee-pager_v2.bin
│   │   └── tee-pager_v2.stm32
│   ├── spl
│   │   └── u-boot-spl.stm32-basic
│   ├── u-boot-basic.img
│   └── u-boot-trusted.stm32
└── ledge-ti-am572x
    ├── MLO -> MLO-ledge-ti-am572x-1.0-r0
    ├── MLO-ledge-ti-am572x -> MLO-ledge-ti-am572x-1.0-r0
    ├── MLO-ledge-ti-am572x-1.0-r0
    ├── optee
    │   ├── tee.bin
    │   ├── tee-header_v2.bin
    │   ├── tee-pageable.bin
    │   ├── tee-pageable_v2.bin
    │   ├── tee-pager.bin
    │   └── tee-pager_v2.bin
    ├── u-boot.img -> u-boot-ledge-ti-am572x-1.0-r0.img
    ├── u-boot-ledge-ti-am572x-1.0-r0.img
    ├── u-boot-ledge-ti-am572x.img -> u-boot-ledge-ti-am572x-1.0-r0.img
```

3.2.4 armv8 family:

```
MACHINE=ledge-multi-armv8 DISTRO=rpb source ./setup-environment build-rpb
bitbake mc:qemuarm64:ledge-iot mc:qemuarm64:ledge-gateway ${FIRMWARE}
```

3.2.5 x86_64:

```
MACHINE=ledge-qemux86-64 DISTRO=rpb source ./setup-environment build-rpb
bitbake ledge-iot ledge-gateway
```

3.3 Install and boot procedure

- DISK="buildid-rootfs.wic" - WIC image generated on build procedure. Like ledge-gateway-ledge-qemuarm64-20200216225638.rootfs.wic.
- OVMF="QEMU_EFI.fd" - OVMF is an EDK II based project to enable UEFI support for Virtual Machines. OVMF contains sample UEFI firmware for QEMU and KVM.

OVMF firmware for different architectures can be downloaded from here: https://storage.kernelci.org/images/uefi/111bbcf87621/

OE maintains script called 'runqemu'. This script automatically added to the path after source ./setup-environment is done. This script can be used to run qemu virtual machine with all required parameters to boot from image and run networking. Configuration file ledge-iot-ledge-qemuarm-*.qemuboot.conf is generated during the build process.

Usage example usage:

```
runqemu ledge-iot-ledge-qemuarm-20200218104425.qemuboot.conf wic serial
```

Example boot log:

```
maxim.uvarov@hackbox2:~/build-test-update/build-rpb-mc/armhf-glibc/deploy/images/
↪ledge-qemuarm$ runqemu ledge-iot-ledge-qemuarm-20200218104425.qemuboot.conf wic␣
↪serial
runqemu - INFO - Running MACHINE=ledge-qemuarm bitbake -e...
runqemu - INFO - Overriding conf file setting of STAGING_DIR_NATIVE to /home/maxim.
↪uvarov/build-test-update/build-rpb-mc/tmp-rpb-glibc/work/armv7at2hf-vfp-linaro-
↪linux-gnueabi/defaultpkgname/1.0-r0/recipe-sysroot-native from Bitbake␣
↪environment
runqemu - INFO - Continuing with the following parameters:

MACHINE: [ledge-qemuarm]
FSTYPE: [wic]
ROOTFS: [/home/maxim.uvarov/build-test-update/build-rpb-mc/armhf-glibc/deploy/
↪images/ledge-qemuarm/ledge-iot-ledge-qemuarm-20200218104425.rootfs.wic]
CONFFILE: [/home/maxim.uvarov/build-test-update/build-rpb-mc/armhf-glibc/deploy/
↪images/ledge-qemuarm/ledge-iot-ledge-qemuarm-20200218104425.qemuboot.conf]

runqemu - INFO - Setting up tap interface under sudo
[sudo] password for maxim.uvarov:
runqemu - INFO - Network configuration: 192.168.7.2::192.168.7.1:255.255.255.0
runqemu - INFO - Using block virtio drive
runqemu - INFO - Interrupt character is '^]'
runqemu - INFO - Running sudo /home/maxim.uvarov/build-test-update/build-rpb-mc/
↪armhf-glibc/work/x86_64-linux/qemu-helper-native/1.0-r1/recipe-sysroot-native/
↪usr/bin/qemu-system-arm -device virtio-net-pci,netdev=net0,mac=52:54:00:12:34:02␣
↪-netdev tap,id=net0,ifname=tap0,script=no,downscript=no -drive id=disk0,file=/
↪home/maxim.uvarov/build-test-update/build-rpb-mc/armhf-glibc/deploy/images/ledge-
↪qemuarm/ledge-iot-ledge-qemuarm-20200218104425.rootfs.wic,if=none,format=raw -
↪device virtio-blk-device,drive=disk0 -no-reboot -show-cursor -device virtio-rng-
↪pci -monitor null -nographic -d unimp -semihosting-config enable,target=native -
↪bios bl1.bin -dtb ledge-qemuarm.dtb -drive id=disk1,file=ledge-kernel-uefi-certs.
↪ext4.img,if=none,format=raw -device virtio-blk-device,drive=disk1  -machine virt,
↪secure=on -cpu cortex-a15 -m 1024  -device virtio-serial-device -chardev null,
↪id=virtcon -device virtconsole,chardev=virtcon

NOTICE:  Booting Trusted Firmware
NOTICE:  BL1: v2.2(debug):v2.2-78-g76f25eb52
NOTICE:  BL1: Built : 08:42:37, Feb 10 2020
INFO:    BL1: RAM 0xe04e000 - 0xe056000
WARNING: BL1: cortex_a15: CPU workaround for 816470 was missing!
INFO:    BL1: cortex_a15: CPU workaround for cve_2017_5715 was applied
```

(continues on next page)

(continued from previous page)

```
INFO:    BL1: Loading BL2
WARNING: Firmware Image Package header check failed.
INFO:    Loading image id=1 at address 0xe01b000
INFO:    Image id=1 loaded: 0xe01b000 - 0xe0201c0
NOTICE:  BL1: Booting BL2
INFO:    Entry point address = 0xe01b000
INFO:    SPSR = 0x1d3
NOTICE:  BL2: v2.2(debug):v2.2-78-g76f25eb52
NOTICE:  BL2: Built : 08:42:37, Feb 10 2020
INFO:    BL2: Doing platform setup
INFO:    BL2: Loading image id 4
WARNING: Firmware Image Package header check failed.
INFO:    Loading image id=4 at address 0xe100000
INFO:    Image id=4 loaded: 0xe100000 - 0xe10001c
INFO:    OPTEE ep=0xe100000
INFO:    OPTEE header info:
INFO:          magic=0x4554504f
INFO:          version=0x2
INFO:          arch=0x0
INFO:          flags=0x0
INFO:          nb_images=0x1
INFO:    BL2: Loading image id 21
WARNING: Firmware Image Package header check failed.
INFO:    Loading image id=21 at address 0xe100000
INFO:    Image id=21 loaded: 0xe100000 - 0xe12e1f8
INFO:    BL2: Skip loading image id 22
INFO:    BL2: Loading image id 5
WARNING: Firmware Image Package header check failed.
INFO:    Loading image id=5 at address 0x60000000
INFO:    Image id=5 loaded: 0x60000000 - 0x600976bc
NOTICE:  BL1: Booting BL32
INFO:    Entry point address = 0xe100000
INFO:    SPSR = 0x1d3


U-Boot 2020.01 (Feb 10 2020 - 08:42:58 +0000)

DRAM:  1 GiB
WARNING: Caches not enabled
Flash: 64 MiB
In:    pl011@9000000
Out:   pl011@9000000
Err:   pl011@9000000
Net:   No ethernet found.
Hit any key to stop autoboot:  0
ERROR: reserving fdt memory region failed (addr=7fe00000 size=200000)
1313 bytes read in 2 ms (640.6 KiB/s)
Scanning disk virtio-blk#30...
Scanning disk virtio-blk#31...
** Unrecognized filesystem type **
Found 4 disks

Warning: virtio-net#32 using MAC address from ROM
ERROR: reserving fdt memory region failed (addr=7fe00000 size=200000)
2299 bytes read in 1 ms (2.2 MiB/s)
ERROR: reserving fdt memory region failed (addr=7fe00000 size=200000)
2299 bytes read in 1 ms (2.2 MiB/s)
Booting: kernel
EFI stub: Booting Linux Kernel...
EFI stub: UEFI Secure Boot is enabled.
EFI stub: Using DTB from configuration table
```

(continues on next page)

(continued from previous page)

```
EFI stub: Exiting boot services and installing virtual address map...
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 5.3.6 (oe-user@oe-host) (gcc version 8.2.1 20180802␣
↪(Linaro GCC 8.2-2018.08~dev)) #1 SMP Tue Feb 18 10:49:14 UTC 2020
[    0.000000] CPU: ARMv7 Processor [412fc0f1] revision 1 (ARMv7), cr=30c5387d
[    0.000000] CPU: div instructions available: patching division code
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, PIPT instruction cache
[    0.000000] OF: fdt: Machine model: linux,dummy-virt
[    0.000000] OF: fdt: Ignoring memory block 0xe00000
```

3.3.1 armv7 (qemu_arm)

```
qemu-system-arm  \
    -device virtio-net-pci,netdev=net0,mac=52:54:00:12:34:02 -netdev tap,id=net0,
↪ifname=tap0,script=no,downscript=no \
    -drive id=disk0,file=${DISK},if=none,format=raw -device virtio-blk-device,
↪drive=disk0 -no-reboot -show-cursor \
    -device virtio-rng-pci -monitor null -nographic \
    -d unimp -semihosting-config enable,target=native -bios bl1.bin -dtb ledge-
↪qemuarm.dtb \
    -drive id=disk1,file=ledge-kernel-uefi-certs.ext4.img,if=none,format=raw -
↪device virtio-blk-device,drive=disk1 \
    -machine virt,secure=on -cpu cortex-a15 -m 1024  -device virtio-serial-device \
    -chardev null,id=virtcon -device virtconsole,chardev=virtcon
```

3.3.2 armv8 (qemu_arm64)

OVMF:

```
qemu-system-aarch64 \
   -cpu cortex-a57 -machine virt -nographic -net nic,model=virtio,
↪macaddr=DE:AD:BE:EF:36:03 -net tap -m 1024 -monitor none \
   -bios ${OVMF} -drive id=disk0,file=${DISK},if=none,format=raw -device virtio-
↪blk-device,drive=disk0 -m 4096 -smp 4 -nographic
```

UBOOT:

```
qemu-system-aarch64 \
   -device virtio-net-pci,netdev=net0,mac=52:54:00:12:34:02 -netdev tap,id=net0,
↪ifname=tap0,script=no,downscript=no \
   -drive id=disk0,file=${ROOTFS},if=none,format=raw -device virtio-blk-device,
↪drive=disk0 -show-cursor \
   -device virtio-rng-pci -monitor null -nographic \
   -d unimp -semihosting-config enable,target=native \
   -bios bl1.bin \
   -drive id=disk1,file=${KEYS},if=none,format=raw \
   -device virtio-blk-device,drive=disk1  -nographic -machine virt,secure=on -cpu␣
↪cortex-a57 -m 4096 -serial mon:stdio -serial null \
   -no-reboot
```

3.3.3 x86_64

```
qemu-system-x86_64 \
   -cpu host -enable-kvm -nographic -net nic,model=virtio,
↪macaddr=DE:AD:BE:EF:36:03 -net tap -m 1024 -monitor none \
```

(continues on next page)

(continued from previous page)

```
    -drive file=${DISK},id=hd,format=raw \
    -drive if=pflash,format=raw,file=${OVMF} \
    -m 4096 -serial mon:stdio -show-cursor -object rng-random,filename=/dev/urandom,
↪id=rng0 -device virtio-rng-pci,rng=rng0
```

3.4 Pre built binaries

Pre built binaries can be downloaded with the following link: http://snapshots.linaro.org/components/ledge/oe/ (Linaro account is required).

CI run task can be found here: https://ci.linaro.org/job/ledge-oe/

CHAPTER

FOUR

DEBIAN

This chapter describes specific Debian LEDGE build and run.

4.1 Supported platfroms

- armv7/armhf
- armv8/arm64

4.2 Build

The Debian build system are based on fai tools (http://fai-project.org) and on a Linaro debian configuration.

Fai tools can be use to generate cross-architecture disk via several way:

1. by using qemu and fai setup

   Described on section ‘Building cross-architecture disk images’ of https://github.com/faiproject/fai/blob/master/doc/fai-guide.txt

2. by using docker image for specific platform

   Only the docker usage are described here.

4.3 Build steps and image generation for armv8

4.3.1 Docker configuration for arm64

Create a Dockerfile for arm64 (Dockerfile.arm64) with following content:

```
FROM arm64v8/debian:buster

# Upgrade system and install fai tools
RUN apt-get update && apt-get -y upgrade && apt-get -y install \
    fai-server fai-setup-storage \
    qemu-utils procps pigz kpartx \
    u-boot-tools dosfstools gdisk

# Clean up APT when done.
RUN apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

# Replace dash with bash
RUN rm /bin/sh && ln -s bash /bin/sh
```

(continues on next page)

(continued from previous page)

```
# Make /home/build the working directory
RUN mkdir -p /home/build
WORKDIR /home/build
```

4.3.2 Script to generate debian content:

Script content (build-fai-linaro-debian.sh):

```
#!/bin/sh
LOCALPATH=$(pwd)
cd fai
sudo fai-diskimage -v --cspace $(pwd) \
     --hostname linaro-ledge-debian \
     -S 4G \
     --class SAVECACHE,BUSTER,DEBIAN,LINARO,LEDGE,RAW \
     "$LOCALPATH"/work.raw
# copy fai log
sudo cp /var/log/fai/linaro-ledge-debian/last/fai.log ../fai.log

# extract debian content from raw
LOOPDEV=$(sudo losetup -f -P --show "$LOCALPATH"/work.raw)
sudo mount $LOOPDEV /mnt/
cd /mnt/
sudo tar cJf "$LOCALPATH"/rootfs-linaro-buster-raw-unknown.ext4 .
cd -
sudo umount /mnt
```

4.3.3 Create docker image:

```
docker build --force-rm --file Dockerfile.arm64 --tag creation .
```

4.3.4 Execute docker image on interactive mode with priviledge:

```
docker run -it --privileged -v $PWD:/home/build creation
```

The current directory are shared with docker.

4.3.5 Create Debian image:

On Docker environment and fai directory

```
DOCKER> ./build-fai-linaro-debian.sh
DOCKER> exit
```

4.3.6 Image generation:

For qemu (arm64):

```
mkdir -p script
wget https://raw.githubusercontent.com/Linaro/meta-ledge/zeus/meta-ledge-bsp/
↪recipes-devtools/generate-raw-image/raw-tools/create_raw_from_flashlayout.sh \
 -O script/create_raw_from_flashlayout.sh
```

(continues on next page)

(continued from previous page)

```
chmod +x script/create_raw_from_flashlayout.sh

wget https://raw.githubusercontent.com/Linaro/meta-ledge/zeus/meta-ledge-bsp/
↪recipes-devtools/generate-raw-image/files/aarch64/FlashLayout_sdcard_arm64_
↪without_boot_firmware.fld -O  FlashLayout_sdcard_arm64_without_boot_firmware.fld

./script/create_raw_from_flashlayout.sh FlashLayout_sdcard_arm64_without_boot_
↪firmware.fld
pigz -9 FlashLayout_sdcard_arm64_without_boot_firmware.raw
```

4.4 Build steps and image generation for armv7

4.4.1 Docker configuration for armhf

Create a Dockerfile for armhf (Dockerfile.armhf) with following content:

```
FROM arm32v7/debian:buster

# Upgrade system and Yocto Proyect basic dependencie
RUN apt-get update && apt-get -y upgrade && apt-get -y install \
    fai-server fai-setup-storage \
    qemu-utils procps pigz kpartx \
    u-boot-tools dosfstools gdisk

# Clean up APT when done.
RUN apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

# Replace dash with bash
RUN rm /bin/sh && ln -s bash /bin/sh

# Make /home/build the working directory
RUN mkdir -p /home/build

# Download fai config
RUN cd /home/build && git clone https://git.linaro.org/ci/fai.git && cd fai && git␣
↪checkout -b WORKING origin/master && cd /home/build

WORKDIR /home/build
```

4.4.2 Script to generate debian content:

Script content (build-fai-linaro-debian.sh):

```
#!/bin/sh
LOCALPATH=$(pwd)
cd fai
sudo fai-diskimage -v --cspace $(pwd) \
     --hostname linaro-ledge-debian \
     -S 4G \
     --class SAVECACHE,BUSTER,DEBIAN,LINARO,LEDGE,RAW \
     "$LOCALPATH"/work.raw
# copy fai log
sudo cp /var/log/fai/linaro-ledge-debian/last/fai.log ../fai.log

# extract debian content from raw
LOOPDEV=$(sudo losetup -f -P --show "$LOCALPATH"/work.raw)
```

(continues on next page)

(continued from previous page)

```
sudo mount $LOOPDEV /mnt/
cd /mnt/
sudo tar cJf "$LOCALPATH"/rootfs-linaro-buster-raw-unknown.ext4 .
cd -
sudo umount /mnt
```

4.4.3 Create docker image:

```
docker build --force-rm --file Dockerfile.armhf --tag creation .
```

4.4.4 Execute docker image on interactive mode with priviledge:

```
docker run -it --privileged -v $PWD:/home/build creation
```

The current directory are shared with docker.

4.4.5 Create Debian image:

On Docker environment and fai directory

```
DOCKER> ./build-fai-linaro-debian.sh
DOCKER> exit
```

4.4.6 Image generation per board:

For qemu (armhf):

```
mkdir -p script
wget https://raw.githubusercontent.com/Linaro/meta-ledge/zeus/meta-ledge-bsp/
↪recipes-devtools/generate-raw-image/raw-tools/create_raw_from_flashlayout.sh \
 -O script/create_raw_from_flashlayout.sh
chmod +x script/create_raw_from_flashlayout.sh

wget https://raw.githubusercontent.com/Linaro/meta-ledge/zeus/meta-ledge-bsp/
↪recipes-devtools/generate-raw-image/files/armv7a/FlashLayout_sdcard_armhf_
↪without_boot_firmware.fld -O FlashLayout_sdcard_armhf_without_boot_firmware.fld

./script/create_raw_from_flashlayout.sh FlashLayout_sdcard_armhf_without_boot_
↪firmware.fld
pigz -9 FlashLayout_sdcard_armhf_without_boot_firmware.raw
```

For stm32mp157c-dk2 (armhf):

```
mkdir -p script
wget https://raw.githubusercontent.com/Linaro/meta-ledge/zeus/meta-ledge-bsp/
↪recipes-devtools/generate-raw-image/raw-tools/create_raw_from_flashlayout.sh \
 -O script/create_raw_from_flashlayout.sh
chmod +x script/create_raw_from_flashlayout.sh

wget https://raw.githubusercontent.com/Linaro/meta-ledge/zeus/meta-ledge-bsp/
↪recipes-devtools/generate-raw-image/files/ledge-stm32mp157c-dk2/FlashLayout_
↪sdcard_ledge-stm32mp157c-dk2-debian.tsv.template -O FlashLayout_sdcard_ledge-
↪stm32mp157c-dk2-debian.tsv.template
```

(continues on next page)

(continued from previous page)

```
./script/create_raw_from_flashlayout.sh FlashLayout_sdcard_ledge-stm32mp157c-dk2-
↪debian.tsv.template
pigz -9 FlashLayout_sdcard_ledge-stm32mp157c-dk2-debian.raw)
```

CHAPTER

FIVE

FIRMWARE

This chapter describes specific firmware generation for LEDGE RP.

CHAPTER

SIX

LEDGE INTERNALS

This chapter discusses specific features for LEDGE RP.

6.1 Applications

ledge-iot image package set is alignment with Fedora IoT package set. List of the packages can be found in bitbake recipe (https://github.com/Linaro/meta-ledge/blob/zeus/meta-ledge-sw/recipes-samples/packagegroups/packagegroup-ledge-iot.bb).

ledge-gateway image includes minimal console image with Ostree and Docker updates support. (https://github.com/Linaro/meta-ledge/blob/zeus/meta-ledge-sw/recipes-samples/images/ledge-gateway.bb)

6.2 U-Boot hardening

Security is very important on EDGE nodes. Hardening and protecting the bootloader is the first step towards a secure system. U-boot command line is disabled and kernel boot parameters are present in LEDGE image.

6.3 QEMU with TF-A and OP-TEE

Ledge image support only qemu versions from 4.1 and above. To load TF-A and OP-TEE with qemu you need to place files in the run directory and name them as bl1.bin, bl2.bin etc:

```
│    ├── bl1.bin -> arm-trusted-firmware/bl1.bin
│    ├── bl2.bin -> arm-trusted-firmware/bl2.bin
│    ├── bl32.bin -> optee/tee-header_v2.bin
│    ├── bl32_extra1.bin -> optee/tee-pager_v2.bin
│    ├── bl32_extra2.bin -> optee/tee-pageable_v2.bin
│    ├── bl33.bin -> u-boot-ledge-qemuarm.bin
```

Then -semihosting and -bios options are used to boot up qemu virtual machine:

```
-d unimp -semihosting-config enable,target=native \
-bios bl1.bin
```

6.4 QEMU with firmware TPM (fTMP) in OP-TEE, TF-A and U-Boot

LEDGE patches default dtb for qemu with ftpm entry.

```
tpm@0 {
        compatible = "microsoft,ftpm";
        linux,sml-base = <0x0 0xC0000000>;
        linux,sml-size = <0x10000>;
};
```

Once qemu is run you might see that tmp_ftpm_tee module privides TEE supplicant storage.

```
~ # rmmod tpm_ftpm_tee && modprobe tpm_ftpm_tee
~ # tpm2_getrandom 256
[  107.057613] audit: type=1130 audit(1574348463.038:33): pid=1 uid=0␣
↪auid=4294967295 ses=4294967295 msg='unit=tpm2-abrmd comm="systemd" exe="/lib/
↪systemd/systemd" hostname=? addr=? terminal=? res=success'
0xD0 0x31 0xA5 0xB9 0xF5 0xD1 0x5D 0x91 0x95 0x19 0x59 0x83 0x9F 0x7D 0x12 0x4F␣
↪0x8F 0xA2 0x8C 0xC2 0x10 0x71 0x09 0x84 0x6F 0x8B 0x1E 0xE6 0xD4 0xA9 0xA8 0xEB␣
↪0xB9 0xAB 0x39 0x92 0x66 0xCB 0x15 0x38 0x7C 0x3F 0x53 0x69 0x86 0xCC 0xA2 0x2A␣
↪0x33 0x6B 0x6D 0xFA 0x62 0xC3 0x70 0x93 0x9F 0x96 0xA8 0xFE 0xDA 0x4B 0x4F 0x15
```

CHAPTER SEVEN

REFERENCES

BIBLIOGRAPHY

[UEFI] Unified Extensable Firmware Interface Specification v2.7A, August 2017, UEFI Forum

INDEX