# EVL Linux Kernel Setup Guide

## EVL

Project page: https://evlproject.org/overview/

A dual kernel architecture. EVL brings real-time capabilities to Linux by embedding a companion core into the kernel, which specifically deals with tasks requiring ultra low and bounded response time to events. For this reason, this approach is known as a dual kernel architecture, delivering stringent real-time guarantees to some tasks alongside rich operating system services to others. In this model, the general purpose kernel and the real-time core operate almost asynchronously, both serving their own set of tasks, always giving the latter precedence over the former.

## Preperation

*$ is host computer, @ is target computer*

- Create an development directory

```
$/> cd ~

$~> mkdir development && cd development
```

- We need gcc/g++ cross compiler for aarch64 and some other utilities

```
sudo apt install \
        gcc-aarch64-linux-gnu g++-aarch64-linux-gnu \
        build-essential git bison flex
```

*Although, these might not be enough*

- First we need to clone the source codes of the EVL/Dovetail patched linux kernel

```
$~/development> git clone --depth 1 --branch evl/v5.10
git://git.evlproject.org/linux-evl.git
```

- Along with it we will also need the libevl sources for the complementary libraries for EVL core.

```
$~/development> git clone --depth 1 --branch master
git://git.evlproject.org/libevl.git
```

...or you could just download the latest relase tag from https://git.evlproject.org/libevl.git

## Compilation

- First start with the kernel

```
$~/development> cd linux-evl

$~/development/linux-evl> make ARCH=arm64 CROSS_COMPILE=aarch64-linux-
gnu- UAPI=~/development/linux-evl defconfig

$~/development/linux-evl> make ARCH=arm64 CROSS_COMPILE=aarch64-linux-
gnu- UAPI=~/development/linux-evl menuconfig
```

- Now when the config menu comes up you will need to change couple of options
  - General Setup **>** Local version `-evl`
  - General Setup **>** Initial RAM filesystem and RAM disk (initramfs/initrd) support `Enable`
  - Platform Selection `disable all except target platform`
  - Kernel Features **>** Dovetail interface `enable`
  - Kernel Features **>** EVL real-time core `enable`
  - CPU Power Management **>** CPU Frequency scaling
    - **>** Default CPUFreq governor `performance`
    - **>** * governor `disable`
    - **>** 'performance' governor `enable`
  - Kernel Hacking **>** Kernel debugging `disable`

### NVIDIA Tegra SoC Specific Options

  - Device Drivers **>** Network device support **>** Ethernet driver support **>** STMicroelectronics devices **>** STMicroelectronics Multi-Gigabit Ethernet driver **>** STMMAC Platform bus support **>** Support for snps,dwc-qos-ethernet.txt DT binding. `module`
  - Device Drivers **>** Thermal drivers **>** NVIDIA Tegra thermal drivers **>** Tegra SOCTHERM thermal management `module`

- Now we are ready to compile the linux kernel

```
$~/development/linux-evl> time make ARCH=arm64 CROSS_COMPILE=aarch64-
linux-gnu- UAPI=~/development/linux-evl -j`nproc`
```

- And continue to building libevl

```
$~/development/linux-evl> cd ../libevl

$~/development/libevl> mkdir ../libevl-build

$~/development/libevl> time make ARCH=arm64 CROSS_COMPILE=aarch64-
```

```
linux-gnu- UAPI=~/development/linux-evl DESTDIR=~/development/libevl-
build install
```

## Installing into target

Now you can use whatever method you desire but for simplicity sake you can just copy `linux-evl/` and `libevl-build/` to target platform. In this guide I have used Jetson AGX Xavier and there will be a platform specific install step at the end of this section.

- Install the kernel

```
@~> cd <target-directory>/linux-evl

@<target-directory>/linux-evl> sudo make modules_install

@<target-directory>/linux-evl> sudo make install
```

- *(Specific for NVIDIA L4T)* Modify boot options

```
@~> sudo vim /boot/extlinux/extlinux.conf
```

and change the contents of this file to the following block. Modify the LINUX and INITRD names according to your build

```
TIMEOUT 30
DEFAULT evl

MENU TITLE L4T boot options

LABEL evl
    MENU LABEL 5.10.0 EVL kernel
    LINUX /boot/vmlinuz-5.10.0-evl-g240f485d65df
    INITRD /boot/initrd.img-5.10.0-evl-g240f485d65df
    APPEND ${cbootargs} isolcpus=1 root=/dev/mmcblk0p1 rw rootwait
rootfstype=ext4 console=ttyTCU0,115200n8 console=tty0 fbcon=map:0
net.ifnames=0

LABEL primary
    MENU LABEL primary kernel
    LINUX /boot/Image
    INITRD /boot/initrd
    APPEND ${cbootargs} quiet root=/dev/mmcblk0p1 rw rootwait
rootfstype=ext4 console=ttyTCU0,115200n8 console=tty0 fbcon=map:0
net.ifnames=0
```

- Install the libevl library

```
@~> cd <target-directory>

@<target-directory>/> sudo mkdir -p /usr/evl

@<target-directory>/> sudo cp -r libevl/* /usr/evl/
```

- Edit .bashrc

```
@~> echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/evl/lib' >>
~/.bashrc

@~> echo 'export PATH=$PATH:/usr/evl/bin' >> ~/.bashrc
```

- *(Specific for NVIDIA L4T)* Flash DTB file

  So, this is pretty easy if you have already flashed the Jetson platform via NVIDIA SDK Manager.

  - Assuming that you have flashed the Jetson with NVIDIA SDK Manager, go into the install directory. It should be under `~/nvidia/.../Linux_for_Tegra`
  - We just have to create a new configuration for the platform we are going to flash the DTB.

    For this example I have used Jetson AGX Xavier for my target platform. Adapt accordingly.

    ```
    cp jetson-xavier.conf jetson-xavier-custom.conf
    echo 'DTB_FILE="tegra194-p2972-0000.dtb";' >> jetson-xavier-
    custom.conf
    ```

    Copy the DTB file from *compiled* linux-evl directory to `kernel/dtb/`

    ```
    cd <~/nvidia/.../Linux_for_Tegra>
    cp ~/development/linux-evl/arch/arm64/boot/dts/nvidia/tegra194-
    p2972-0000.dtb kernel/dtb
    ```

  - Flash the target

    First put the target into Force Recovery Mode. For Jetson Xavier, this is accomplished by pressing the middle button on the button group that is found on one side of the device. *While pressing that button* turn on the device and release all buttons.

    ```
    sudo ./flash.sh -r -k kernel-dtb jetson-xavier-custom mmcblk0p1
    ```

## Finish EVL Setup

```
# Check kernel setup
evl test
evl check

# Tune real-time core
latmus -t

# Stress test + latency measurement
hectic &
latmus -m
```

If you are observing anormal latency figures then it could be a result of improper value of `isolcpus` on kernel commandline. One another problem could be the result of imporper scaling_governor on cpus. You can check this by executing following command

```
cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

This should print only `performance`.