

Menuconfig or how to configure kernel

Stable: 03.05.2019 - 16:50 / Revision: 02.05.2019 - 18:19

Contents

1 Linux configuration genericity	1
2 Menuconfig and Developer Package	3
3 Menuconfig and Distribution Package	5
4 References	5

1 Linux configuration genericity

The process of building a kernel has two parts: configuring the kernel options and building the source with those options.

The Linux[®] kernel configuration is found in the generated file: `.config`.

`.config` is the result of configuring task which is processing platform `defconfig` and fragment files if any.

For OpenSTLinux distribution the `defconfig` is located into the kernel source code and fragments into `stm32mp` BSP layer :

- `arch/arm/configs/multi_v7_defconfig`

Every new kernel version brings a bunch of new options, we do not want to back port them into a specific `defconfig` file each time the kernel releases, so we use the same `defconfig` file based on ARM SoC v7 architecture.

STM32MP1 specificities are managed with fragments config files.

- `meta-st/meta-st-stm32mp/recipes-kernel/linux/linux-stm32mp/<kernel version>/fragment-*.config`

`.config` result is located in the build folder:

- `build-openstlinuxweston-stm32mp1/tmp-glibc/work/stm32mp1-openstlinux_weston-linux-gnueabi/linux-stm32mp/4.14-48/linux-stm32mp1-standard-build/.config`

To modify the kernel options, it is not recommended to edit this file directly.

- A user runs either a text-mode :

```
PC $> make config
starts a character based question and answer session (Figure 1)
```

```
[greg@shamp linux-2.5]$ make config
make[1]: `scripts/kconfig/conf' is up to date.
./scripts/kconfig/conf arch/i386/Kconfig
#
# using defaults found in .config
#
*
* Linux Kernel Configuration
*
*
* Code maturity level options
*
Prompt for development and/or incomplete code/drivers (EXPERIMENTAL) [Y/n/?]
```

Figure 1. Configuring the kernel with make config

```
Linux Kernel v2.5.59 Configuration

Linux Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help.
Legend: [*] built-in [ ] excluded <M> module < > module capable

Code maturity level options --->
General setup --->
Loadable module support --->
Processor type and features --->
Power management options (ACPI, APM) --->
Bus options (PCI, PCMCIA, EISA, MCA, ISA) --->
Executable file formats --->
Memory Technology Devices (MTD) --->
Parallel port support --->
Plug and Play support --->

<Select> < Exit > < Help >
```

Figure 2. Make menuconfig makes it easier to back up and correct mistakes

mid level driver) follow. Only one of these should appear in an actual .config file:

```
CONFIG_SCSI=y
CONFIG_SCSI=m
# CONFIG_SCSI is not set
```

PC \$> make menuconfig starts a terminal-oriented ncurses text version of the kernel configuration. [Wikipedia Menuconfig](#)

- or a graphical kernel configurator :

PC \$> make xconfig starts a X based configuration tool.

Ultimately these configuration tools edit the .config file.

An option indicates either some driver is built into the kernel ("=y") or will be built as a module ("=m") or is not selected.

The unselected state can either be indicated by a line starting with "#" (e.g. "# CONFIG_SCSI is not set") or by the absence of the relevant line from the .config file.

The 3 states of the main selection option for the SCSI subsystem (which actually selects the SCSI

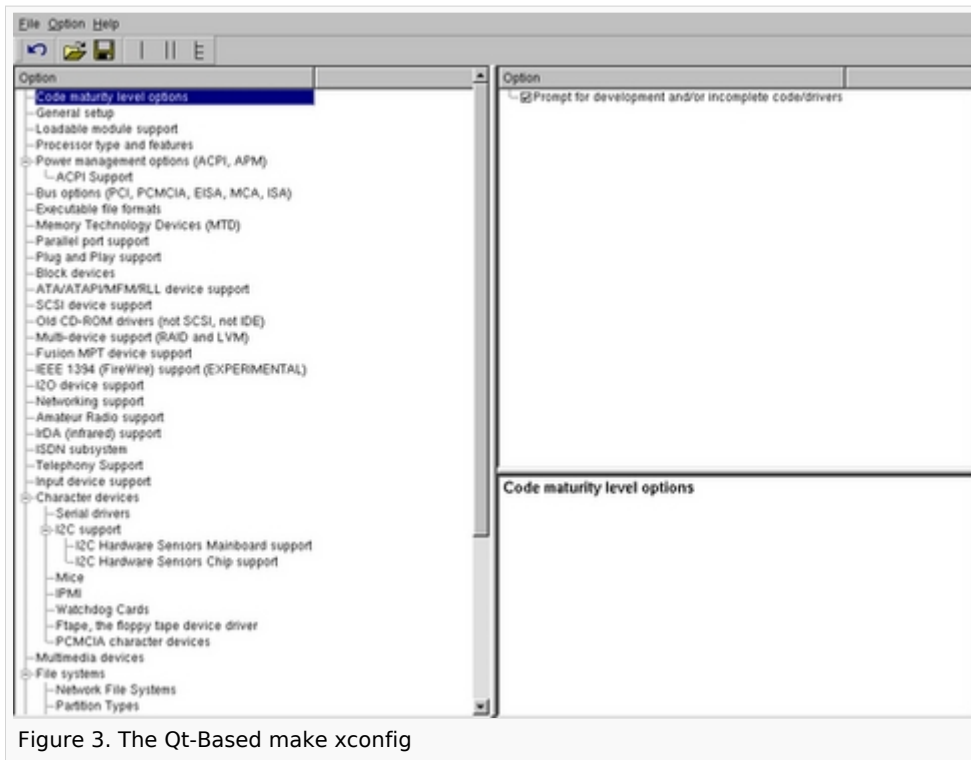


Figure 3. The Qt-Based make xconfig

2 Menuconfig and Developer Package

For this use case, the prerequisite is that OpenSTLinux SDK has been installed and configured.

To verify if your cross-compilation environment has been put in place correctly, run the following command:

```
PC $> set | grep CROSS
CROSS_COMPILE=arm-openstlinux_weston-linux-gnueabi-
```

For more details, refer to *<Linux kernel installation directory>/README.HOW_TO.txt* helper file (the latest version of this helper file is also available in this user guide: [README.HOW_TO.txt](#)).

- Go to the *<Linux kernel build directory>*

```
PC $> cd <Linux kernel build directory>
```

- Save initial configuration (to identify later configuration updates)

```
PC $> make arch=ARM savedefconfig
Result is stored in defconfig file
PC $> cp defconfig defconfig.old
```

- Start the Linux kernel configuration menu

```
PC $> make arch=ARM menuconfig
```

- Navigate forwards or backwards directly between feature
 - un/select, modify feature(s) you want
 - When the configuration is OK : exit and save the new configuration

```
useful keys to know:  
enter: enter in config subdirectory  
space: hit several times to either select [*], select in module [m] or unselect [ ]  
/: to search for a keyword, this is usefull to navigate in tree  
?: to have more information on selected line
```

- Compare the old and new config files after operating modifications with menuconfig

```
PC $> make arch=ARM savedefconfig
```

Retrieve configuration updates by comparing the new defconfig and the old one

```
PC $> meld defconfig defconfig.old
```

- Cross-compile the Linux kernel (please check the load address in the *README.HOW_TO.txt* helper file)

```
PC $> make arch=ARM uImage LOADADDR=<loadaddr of kernel>  
PC $> cp arch/arm/boot/uImage install_artifact/boot/
```

- Update the Linux kernel image on board

```
PC $> scp install_artifact/boot/uImage root@<board ip address>:/boot/
```



If the */boot* mounting point doesn't exist yet, please see [how to create a mounting point](#)

- Reboot the board

```
Board $> cd /boot; sync; systemctl reboot
```

Note that this use case modifies the configuration file in the Linux kernel build directory, not in the Linux kernel source directory: this is a temporary modification useful for a prototyping.

- To make this temporary modification permanent, the delta between defconfig and defconfig.old must be saved in a configuration fragment file (fragment-*.config) based on fragment.cfg file, and the Linux kernel configuration/compilation steps must be re-executed (as explained in the *README.HOW_TO.txt* helper file).

3 Menuconfig and Distribution Package

- Start the Linux kernel configuration menu

```
PC $> bitbake virtual/kernel -c menuconfig
```

- Navigate forwards or backwards directly between feature
 - un/select, modify feature(s) you want
 - When the configuration is OK : exit and save the new configuration

useful keys to know:

enter: enter in config subdirectory

space: hit several times to either select **[*]**, select in module **[m]** or unselect **[]**

/: to search for a keyword, this is usefull to navigate in tree

?: to have more information on selected line

- Cross-compile the Linux kernel

```
PC $> bitbake virtual/kernel
```

- Update the Linux kernel image on board

```
PC $> scp <build dir>/tmp-glibc/deploy/images/<machine name>/uImage root@<board ip address>
```



If the `/boot` mounting point does not exist yet, please see [how to create a mounting point](#)

- Reboot the board

```
Board $> cd /boot; sync; systemctl reboot
```

Note that this use case modifies the configuration file in the Linux kernel build directory, not in the Linux kernel source directory: this is a temporary modification useful for a prototyping.

- To make this temporary modification permanent, it must be saved in a configuration fragment file (fragment-*.config) based on **fragment.cfg** file, and the Linux kernel configuration/compilation steps must be re-executed: **bitbake <name of kernel recipe>**.

4 References

1. [↑ Wikipedia Menuconfig](#)

Board Support Package

Board support package

Software development kit

