ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

Configfs

For further information contact your local STMicroelectronics sales office.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

Contents

1. Configfs 3
2. Menuconfig or how to configure kernel 7
3. USB overview 13

A quality version of this page, approved on *3 February 2020*, was based off this revision.

Contents

1 Introduction 4
2 Installing configs on your target board 5
3 Getting started 6
3.1 How to mount configfs 6
3.2 How to set and manage configfs from Linux kernel drivers and user space 6
4 References 7

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

1 Introduction

Configfs[1] is a RAM-based filesystem that provides the converse of sysfs functionality.

While sysfs provides a filesystem-based view of kernel objects, configfs is a filesystem-based manager of kernel objects or config_items (every object in configfs is a config_item). This means that kernel objects can be created, managed and destroyed from the user space.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

2 Installing configs on your target board

Configfs can be enabled and ready to be used in all STM32MPU Embedded Software distribution, via the Linux® kernel configuration **CONFIG_CONFIGFS_FS** (set to yes by default):

```
Symbol: CONFIGFS_FS
Location:
  File systems --->
    Pseudo filesystems -->
      -*- Userspace-driven configuration filesystem
```

Please refer to Menuconfig or how to configure kernel article for instructions for modifying the configuration and recompiling the Linux kernel image in the Distribution Package context.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

3 Getting started

3.1 How to mount configfs

Use the following command to mount **Configfs** at */sys/kernel/config*:

```
Board $> mount -t configfs none /sys/kernel/config
```

3.2 How to set and manage configfs from Linux kernel drivers and user space

Refer to the Linux documentation[1] for detailed information.

Configfs is used by the USB framwework. Refer to USB API description for an example.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

4 References

- 1.01.1 Documentation/filesystems/configfs/configfs.txt

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)

System File System (See https://en.wikipedia.org/wiki/Sysfs for more details)

Configuration File System (See https://en.wikipedia.org/wiki/Configfs for more details)

Linux® is a registered trademark of Linus Torvalds.

Stable: 11.02.2021 - 11:10 / Revision: 19.01.2021 - 10:34

A quality version of this page, approved on *11 February 2021*, was based off this revision.

Contents

1 Linux configuration genericity 8
2 Menuconfig and Developer Package 10
3 Menuconfig and Distribution Package 12
4 References 13

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-ostl-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

```
PC $> make menuconfig
starts a terminal-oriented configuration tool (using ncurses) (Figure 2)
The ncurses text version is more popular and is run with the make menuconfig option.
Wikipedia Menuconfig[1] also explains how to "navigate" within the configuration menu, and highlights main key strokes.
```

- or a graphical kernel configurator :

```
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

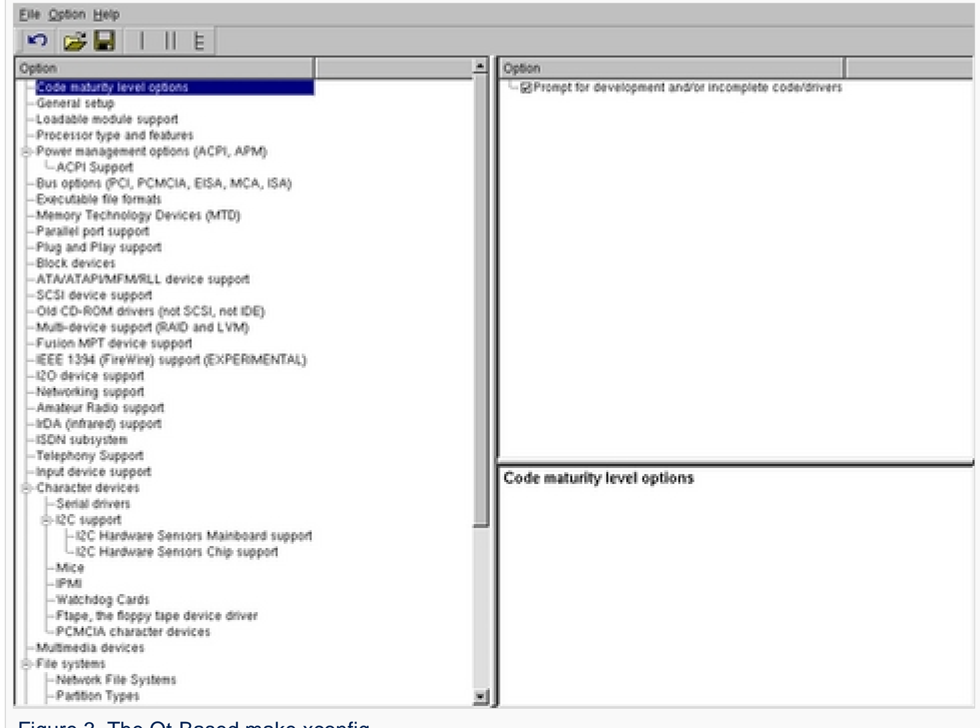

Figure 3. The Qt-Based make xconfig

```
PC $> make xconfig
starts a X based
configuration tool
(Figure 3)
```

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 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
```

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

2 Menuconfig and Developer Package

For this use case, the prerequesite 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-ostl-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 GitHub: 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)

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

```
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/
```

Information

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).

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

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>:/boot
```

Information

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>**.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

4 References

- Wikipedia Menuconfig

Linux® is a registered trademark of Linus Torvalds.

Board support package

Software development kit (A programming package that enables a programmer to develop applications for a specific platform.)

Stable: 16.02.2021 - 16:16 / Revision: 16.02.2021 - 15:49

A quality version of this page, approved on *16 February 2021*, was based off this revision.

This article gives information about the Linux® USB framework.
It explains how to activate USB interface and, based on examples, how to access it from user space.

Contents

1 Framework purpose 14
2 System overview 15
2.1 Component description 16
2.2 API description 16
3 Configuration 17
3.1 Kernel configuration 17
3.2 Device tree configuration 17
4 How to use the framework 18
4.1 How to list USB devices 18
4.2 How to mount a USB key (mass-storage) 18
4.3 How to configure USB Gadget through configfs 19
5 How to trace and debug the framework 20
5.1 How to monitor 20
5.1.1 How to monitor with debugfs 20
5.1.2 How to monitor with sysfs 21
5.1.2.1 USB buses monitoring with sysfs 21
5.1.2.2 USB Gadget monitoring with sysfs 21
5.2 How to trace 21
5.2.1 How to trace with usbmon 21
5.2.2 How to trace using a protocol analyzer 22
5.3 How to debug 22
5.3.1 Activating USB framework debug messages 22
5.3.2 EHCI/OHCI driver debugfs entry 22
5.3.3 DWC2 driver debug messages and debugfs entry 22
6 Source code location 24
7 References 25

ST Restricted – Subject to Non-Disclosure Agreement – Do not copy

1 Framework purpose

The USB (universal serial bus) Linux® framework supports many types of:

- host controllers and peripheral devices
- gadget drivers and classes to be used within a peripheral

Linux can be used on the host machine. In this case various types of peripherals can be plugged in, such as:

- Mass storage (hard drive, USB stick..)
- HID (keyboard, mouse..)

Linux can also be used as a device on the peripheral side, using gadget drivers. In this case, it can act as:

- USB mass storage (e.g. to export some partitions, filesystem)
- ethernet card
- serial interface
- ...

2 System overview

Legend
3rd Party
ST
Community
Community + ST adds-on
Hardware
Direct link
Indirect link (via other frameworks)
Interface

User terminal
User applications
usbutils
libusbg API
libusb API
libusbg
libusb
debugfs
sysfs
configfs
block device
char device
User space
Kernel space
USB user drivers
USB gadget driver
USB class driver
USB core
Peripheral core
Host core
USB controller drivers
OTG controller driver
Host controller driver
PHY
PHY core
PHY driver
STM32
Hardware
OTG
USBH
UTMI+
UTMI+
Dedicated FS PHY
USBPHYC
USB bus
USB bus
USB bus
External devices
"A-device" or "B-device"
"A-device" or "B-device"
"B-device"

USB Implementation architecture

ST Restricted – Subject to Non-Disclosure Agreement – Do not copy

2.1 Component description

- **USB userland** (*User space*)
  - *Host-Side* userland

  - libusb[1] is a userland library that provides access to USB devices.
  - usbutils[2] is a set of USB utilities for collecting information about the USB devices that are connected to the USB host. Note that **usbutils** depends on **libusb**.
  - One of the well-known utility is **lsusb**, used to display information about USB buses and the devices connected to them.

  - *Gadget* userland

  - libusbg[3] is a userland library that provides routines for creating and parsing USB gadget devices using the configfs API.
  - Gadget configfs provides configuration interface available through user terminal, used to configure USB Gadget.

  - Common userland

  - **sysfs** provides an information interface available through the user terminal. See How to monitor with sysfs below.
  - **debugfs** provides a debugging interface available through the user terminal. See How to monitor with debugfs below.

- **USB framework** (*Kernel space*): composed of two parts, USB *Host-Side* and USB *Gadget*, which rely on the USB core with specific APIs to support USB host and devices controllers
  - Host-Side provides API interface to class drivers and forwards the request from class drivers to host controller driver.
  - Gadget requires a peripheral controller and the gadget driver to use it.
- **USB controller drivers** (*Kernel space*)
  - *USB Host controller drivers* such as STM32 USBH USB Host controllers in the USB Host-Side framework. STM32 USBH uses kernel community drivers (kernel space), based on the USB framework.

  - Enhanced Host Controller Interface (EHCI) driver and Generic platform ehci driver
  - Open Host Controller Interface (OHCI) driver and Generic platform ohci driver

  - *USB OTG controller drivers* such as STM32 OTG USB OTG controllers in the USB Host-Side framework when they are used either in **otg** or **host** mode, and/or in the USB Gadget framework when they are used either in **otg** or **peripheral** mode. STM32 OTG uses kernel community driver (kernel space), based on the USB framework.

  - DesignWare HS OTG Controller driver

  - USB controller drivers can rely on Generic PHY framework to manage the physical layer for USB data transmissions. STM32 USBPHYC PHY provider is a *PHY driver* in the Generic PHY framework:

  - STM32 USBPHYC driver

- **USB hardware controllers** (*Hardware*)

USB controllers such as STM32 USBH internal peripheral and STM32 OTG internal peripheral, using an on-chip High-Speed UTMI+ PHY (STM32 USBPHYC internal peripheral), or on-chip Full-Speed PHY for STM32 OTG internal peripheral.

- **USB devices** (*External USB devices*)
  - USB OTG specification[4] defines two roles for USB devices: A-Device and B-Device. STM32 OTG controller, depending on the USB connector which is used, can accept both A-Device and B-Device, while STM32 USBH Host controller only manages B-Device:

  - **A-Device** is a **power supplier** acting as a **USB Host** (e.g. a PC)
  - **B-Device** is a **power consumer** and acts as a **USB Peripheral** (e.g. a USB key).

2.2 API description

See USB kernel documentation for more details on API functions.

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

3 Configuration

3.1 Kernel configuration

USB support, STM32 USBH driver and STM32 OTG driver are activated by default in ST deliveries. Nevertheless, if a specific configuration is required, this section indicates how the USB framework can be activated/deactivated in the kernel.

Activate USB support (CONFIG_USB=y) in the kernel configuration with the Linux Menuconfig tool: Menuconfig or how to configure kernel then select:

```
Device Drivers  --->
 [*] USB support  --->
```

Then activate USB controllers drivers.

To activate the STM32 USBH driver, select:

```
Device Drivers  --->
   --- USB support
   <*>   Support for Host-side USB
   <*>   EHCI HCD (USB 2.0) support
   <*>     Generic EHCI driver for a platform device
   <*>   OHCI HCD (USB 1.1) support
   <*>     Generic OHCI driver for a platform device
```

To activate the STM32 OTG driver, select:

```
Device Drivers  --->
   --- USB support
   <*>   Support for Host-side USB
   <*>   USB Gadget Support  --->
   <*>   DesignWare USB2 DRD Core Support
           DWC2 Mode Selection (Dual Role mode)  --->
```

Then to activate the STM32 USBPHYC driver, select:

```
PHY Subsystem  --->
  -*- PHY Core
  <*> STMicroelectronics STM32 USB HS PHY Controller driver
```

3.2 Device tree configuration

Detailed DT configurations for STM32 USB internal peripherals:

- for STM32 USBH Host controller: USBH device tree configuration
- for STM32 OTG controller: OTG device tree configuration
- for STM32 USBPHYC PHY: USBPHYC device tree configuration

4 How to use the framework

4.1 How to list USB devices

lsusb displays information about the attached USB buses and devices.

In the example above, we have an onboard hub, a USB mouse and USB keyboard plugged into the hub.

```
Board $> lsusb                                                /* root hubs
correspond to STM32 USB controllers (USBH,  OTG) */
Bus 002 Device 005: ID 413c:2003 Dell Computer Corp. Keyboard
Bus 002 Device 004: ID 046d:c016 Logitech, Inc. Optical Wheel Mouse
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 002: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

```
Board $> lsusb -t                                              /* lsusb -t shows
the USB class, the driver used and the number of ports and speed of each USB devices */
/:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=ehci-platform/2p, 480M
    |__ Port 1: Dev 2, If 0, Class=Hub, Driver=hub/4p, 480M
        |__ Port 1: Dev 5, If 0, Class=Human Interface Device, Driver=usbhid, 1.5M
        |__ Port 3: Dev 4, If 0, Class=Human Interface Device, Driver=usbhid, 1.5M
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=dwc2/1p, 480M
```

To limit **lsusb** to the USB keyboard:

```
Board $> lsusb -s 002:005                                   /* lsusb -s [Bus]:
[Device] */
Bus 002 Device 005: ID 413c:2003 Dell Computer Corp. Keyboard
Board $> lsusb -d 413c:2003                                     /* lsusb -d [ID] */
Bus 002 Device 005: ID 413c:2003 Dell Computer Corp. Keyboard
```

To limit **lsusb** to the USB keyboard and display its descriptors:

```
Board $> lsusb -D /dev/bus/usb/002/005                      /* lsusb -D /dev/bus/usb/
[Bus]/[Device] */
Device: ID 413c:2003 Dell Computer Corp. Keyboard
Device Descriptor:
 ...
```

4.2 How to mount a USB key (mass-storage)

```
Board $> mkdir /usb
Board $> mount /dev/sdxx /usb
```

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

4.3 How to configure USB Gadget through configfs

See USB Gadget configfs documentation for an introduction to USB gadget configfs structure and how to use it to configure Linux USB Gadget.

Here is an example to configure USB Gadget through configfs to use the OTG as a USB Ethernet Gadget with Remote NDIS (RNDIS). See: stm32_usbotg_eth_config.sh .

5 How to trace and debug the framework

5.1 How to monitor

5.1.1 How to monitor with debugfs

Please refer to the USB devices chapter[5] to decode the output.

```
Board $> cat /sys/kernel/debug/usb/devices

T:  Bus=01 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=480  MxCh= 1
B:  Alloc=  0/800 us ( 0%), #Int=  0, #Iso=  0
D:  Ver= 2.00 Cls=09(hub  ) Sub=00 Prot=01 MxPS=64 #Cfgs=  1
P:  Vendor=1d6b ProdID=0002 Rev= 4.14
S:  Manufacturer=Linux 4.14.0 dwc2_hsotg
S:  Product=DWC OTG Controller
S:  SerialNumber=49000000.usb-otg
C:* #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=  0mA
I:* If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   4 Ivl=256ms

T:  Bus=01 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  4 Spd=480  MxCh= 0
D:  Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=05e3 ProdID=0723 Rev=94.54
S:  Manufacturer=Generic
S:  Product=USB Storage
C:* #Ifs= 1 Cfg#= 1 Atr=80 MxPwr=500mA
I:* If#= 0 Alt= 0 #EPs= 2 Cls=08(stor.) Sub=06 Prot=50 Driver=usb-storage
E:  Ad=81(I) Atr=02(Bulk) MxPS= 512 Ivl=0ms
E:  Ad=02(O) Atr=02(Bulk) MxPS= 512 Ivl=0ms

T:  Bus=02 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=480  MxCh= 2
B:  Alloc=  0/800 us ( 0%), #Int=  2, #Iso=  0
D:  Ver= 2.00 Cls=09(hub  ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=1d6b ProdID=0002 Rev= 4.14
S:  Manufacturer=Linux 4.14.0 ehci_hcd
S:  Product=EHCI Host Controller
S:  SerialNumber=5800d000.usbh-ehci
C:* #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=  0mA
I:* If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   4 Ivl=256ms

T:  Bus=02 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=480  MxCh= 4
D:  Ver= 2.00 Cls=09(hub  ) Sub=00 Prot=02 MxPS=64 #Cfgs=  1
P:  Vendor=0424 ProdID=2514 Rev= b.b3
C:* #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=  2mA
I:  If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=01 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   1 Ivl=256ms
I:* If#= 0 Alt= 1 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=02 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   1 Ivl=256ms

T:  Bus=02 Lev=02 Prnt=02 Port=03 Cnt=01 Dev#=  5 Spd=1.5  MxCh= 0
D:  Ver= 1.10 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=413c ProdID=2003 Rev= 1.00
S:  Manufacturer=Dell
S:  Product=Dell USB Keyboard
C:* #Ifs= 1 Cfg#= 1 Atr=a0 MxPwr= 70mA
I:* If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=01 Prot=01 Driver=usbhid
E:  Ad=81(I) Atr=03(Int.) MxPS=   8 Ivl=24ms
```

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

5.1.2 How to monitor with sysfs

5.1.2.1 USB buses monitoring with sysfs

Please refer to What are the sysfs structures for Linux USB?[6].

```
Board $> ls /sys/bus/usb/devices/
1-0:1.0  1-1  1-1:1.0  2-0:1.0  2-1  2-1.4  2-1.4:1.0  2-1:1.0  usb1  usb2
```

The names that begin with **usb** refer to USB controllers.

The device naming scheme is the following:

- bus-port.port.port... (**1-1**, **2-1**, or **2-1.4** in the example above)

The interfaces are indicated by suffixes in the following form:

- :config.interface (1-1**:1.0**, 2-1**:1.0**, 2-1.4**:1.0** in the example above)

Each interface corresponds to an entry in sysfs and can have its own driver.

5.1.2.2 USB Gadget monitoring with sysfs

Once the USB Gadget is configured, USB Device Controller sysfs is populated. See Documentation/ABI/stable/sysfs-class-udc for a description of each file.

```
Board $> ls /sys/class/udc/49000000.usb-otg/
a_alt_hnp_support  device           is_selfpowered  srp
a_hnp_support      function         maximum_speed   state
b_hnp_enable       is_a_peripheral  power           subsystem
current_speed      is_otg           soft_connect    uevent
```

5.2 How to trace

5.2.1 How to trace with usbmon

usbmon [7] collects traces of the input/output on the USB bus.
It relies on a kernel part and on a user part, and reports the requests made by USB device drivers to the Host controller drivers.
Activate USBMON support (CONFIG_USB_MON=y) in the kernel configuration with Linux Menuconfig tool: Menuconfig or how to configure kernel.
A usbmon entry is created in debugfs. It includes several files.
The file names consist of a number (the USB bus - 0 relates to all buses) and a letter (s, u or t). The s file contains a generic event overview. The t (deprecated) and u files will stream trace data.
To gather debug data, either use the master file 0u (to capture data from all devices) or find out the bus to which your device is connected and use the corresponding bus file. For example, if the device is connected to bus 1:

```
Board $> cat /sys/kernel/debug/usb/usbmon>1u > bus1data.log
```

To stop the capture, just type (CTRL+C) to kill the command. You can then analyze the log with vUSBAnalyzer graphical tool on your Linux host.

5.2.2 How to trace using a protocol analyzer

A USB protocol analyzer is a USB traffic sniffer that decodes USB descriptors and displays bus states and packets sent. Refer to you USB protocol analyzer user manual.

5.3 How to debug

5.3.1 Activating USB framework debug messages

A detailed dynamic trace is available in How to use the kernel dynamic debug

```
Board $> echo  "file usb* +p" > /sys/kernel/debug/dynamic_debug/control
```

This command enables all the traces related to the USB core and drivers at runtime.
A finer selection can be made by choosing only the files to trace.

Information

Reminder: *loglevel* needs to be increased to 8 either by using boot arguments or by sending the *dmesg -n 8* command from the console

5.3.2 EHCI/OHCI driver debugfs entry

EHCI/OHCI drivers export a debugfs entry when CONFIG_DYNAMIC_DEBUG is enabled.

```
Board $> ls /sys/kernel/debug/usb/ohci/5800c000.usbh-ohci/
async  periodic  registers
Board $> ls /sys/kernel/debug/usb/ehci/5800d000.usbh-ehci/
async  bandwidth  periodic  registers
```

- **async** dumps a snapshot of the async schedule.
- **bandwith** dumps the bandwidth allocation
- **periodic** dumps a snapshot of the periodic schedule.
- **registers** dumps the USB controller registers

5.3.3 DWC2 driver debug messages and debugfs entry

To get the verbose messages from the DWC2 driver used by STM32 OTG, activate "Enable Debugging Messages" in the Linux kernel via the menuconfig Menuconfig or how to configure kernel.

```
Device Drivers  --->
   [*] USB support
   <*>   Support for Host-side USB
   <*>   USB Gadget Support  --->
   <*>   DesignWare USB2 DRD Core Support
   [*]     Enable Debugging Messages
   [*]       Enable Verbose Debugging Messages
   [ ]     Enable Missed SOF Tracking
   [*]     Enable Debugging Messages For Periodic Transfers
```

This can be done manually in your kernel .config file:

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

```
CONFIG_USB_SUPPORT=y
CONFIG_USB_DWC2=y
CONFIG_USB_DWC2_DEBUG=y
CONFIG_USB_DWC2_VERBOSE=y
CONFIG_USB_DWC2_DEBUG_PERIODIC=y
```

The debug support for DWC2 driver (CONFIG_USB_DWC2_DEBUG) compiles all the files located in Linux kernel drivers/usb/dwc2/ folder with DEBUG flag.

Information

Reminder: *loglevel* needs to be increased to 8 by using either boot arguments or the *dmesg -n 8* command through the console

The DWC2 driver also exports a debugfs entry that contains useful information:

```
Board $> ls /sys/kernel/debug/49000000.usb-otg/
dr_mode  ep0  ep1in  ep1out  ep2in  ep2out  ep3in  ep3out  ep4in  ep4out  ep5in  ep5out
ep6in  ep6out  ep7in  ep7out  ep8in  ep8out  fifo  hw_params  params  regdump state
testmode
```

- **dr_mode** indicates the working mode of the USB controller. It can be "host", "peripheral" or "otg". The value is set through a device tree property.
- **ep*** files show the state of the given endpoint.
- **fifo** shows the FIFO information for the overall FIFO and all the periodic transmission FIFOs.
- **hw_params** shows the parameters read from USB controller registers.
- **params** shows the parameters used by the driver.
- **regdump** dumps all the USB controller registers.
- **state** shows the overall state of the hardware controller and some general information on the available endpoints.
- **testmode** shows/sets usb test mode ("test_j", "test_k", "test_se0_nak", "test_packet", "test_force_enable").

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

6 Source code location

The source files are located inside the Linux kernel.

- The **USB framework** is under drivers/usb/
- The drivers used for STM32 USBH are under drivers/usb/host/ehci-platform.c , drivers/usb/host/ehci-hcd.c and drivers/usb /host/ohci-platform.c , drivers/usb/host/ohci-hcd.c
- The driver used for STM32 OTG is under drivers/usb/dwc2/

ST Restriced – Subject to Non-Disclosure Agreement – Do not copy

7 References

- libusb: a cross-platform library to access USB devices
- usbutils: USB utilities for Linux, including lsusb
- libusbg: a C library encapsulating the kernel USB gadget-configfs userspace API functionality
- On-The-Go and Embedded Host Supplement to the USB Revision 2.0 Specification
- Linux USB API: The Linux-USB Host Side API - The USB devices
- What are the sysfs structures for Linux USB?
- usbmon

Linux® is a registered trademark of Linus Torvalds.

Human Interface Device (for USB, Bluetooth...)

Configuration File System (See https://en.wikipedia.org/wiki/Configfs for more details)

Application programming interface

System File System (See https://en.wikipedia.org/wiki/Sysfs for more details)

Debug File System (See https://en.wikipedia.org/wiki/Debugfs for more details)

USB On-The-Go (Capability/type of USB port, acting primarily as USB device, to also act as USB host. Also known as USB OTG.)

USB 2.0 Transceiver Macrocell Interface

Enhanced Host Controller Interface

Open Host Controller Interface

Dual-Role Device (USB standard defines host and device roles. OTG controllers support both roles and can be called Dual-Role Devices controllers.)

High Speed (MIPI® Alliance DSI standard)

Device Tree