



life.augmented

Developer package hands on

Cindy Ge

March 2021

Agenda

1 Prerequisites

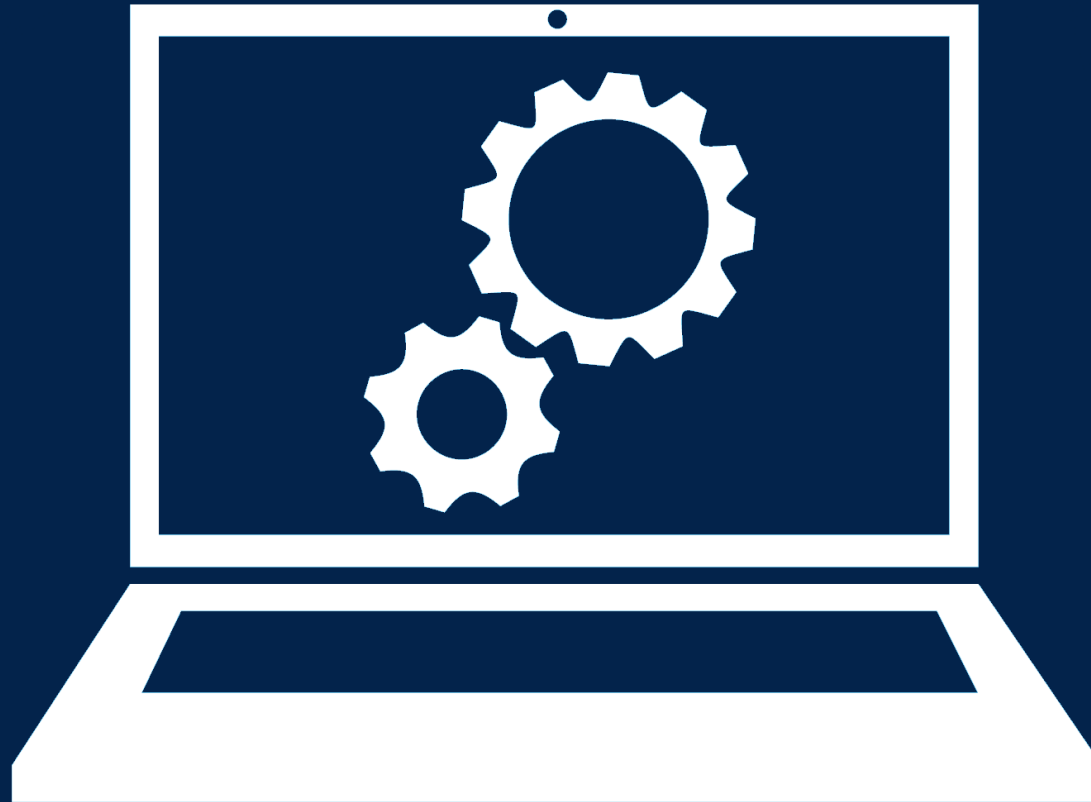
2 TF-A Development

3 U-Boot Development

4 Linux Kernel Development

5 STM32CubeMX & FDCAN Demo

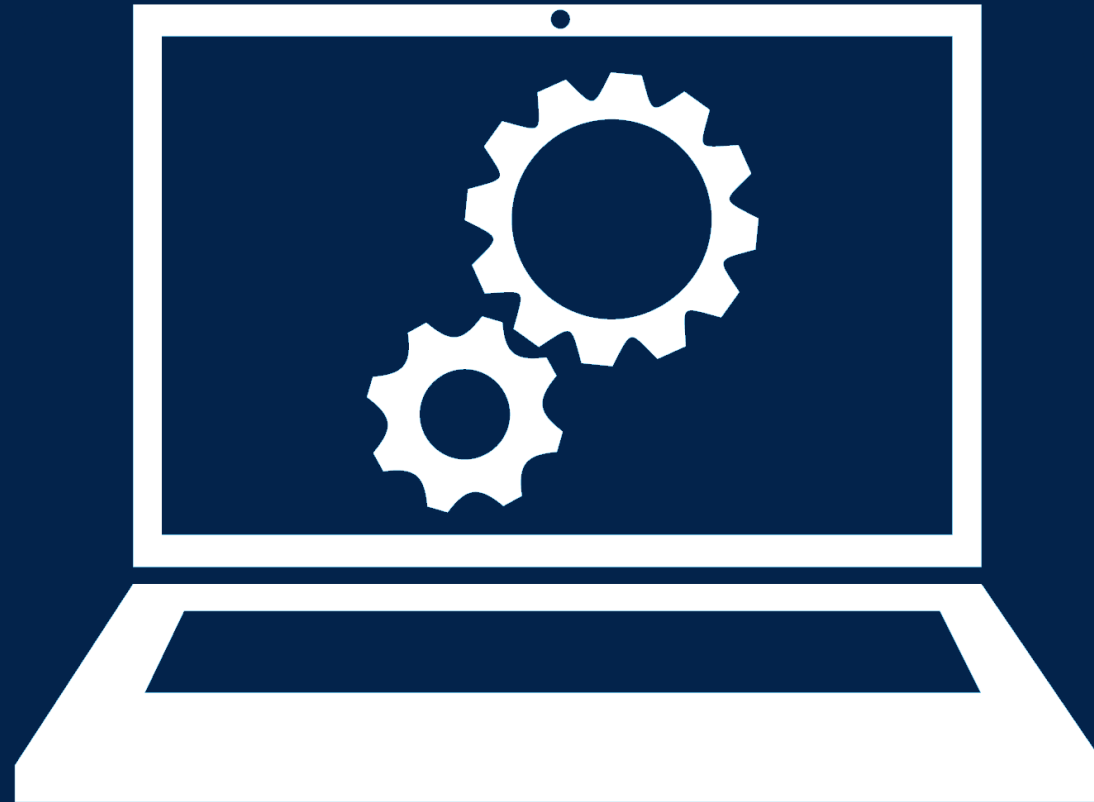
Prerequisites



Prerequisites

- A STM32MP1 board with some standard connector (USB)
- A standard PC running in priority:
 - Ubuntu LTS (18.04 or 20.04)
 - Windows PC able to run VMWare
 - At least 4 cores, 100GB free disk and 8GB DDR is recommended
- Developer package has been installed
- STM32CubeMX has been installed
- FDCAN transceiver modules

TF-A Development



TF-A Build Flags

Mandatory flags:

- ***ARM_ARCH_MAJOR=7***: the major version of ARM Architecture to target (STM32MP15 is ARMv7 architecture based)
- ***ARCH=aarch32***: specify aarch32 architecture to be built
- ***PLAT=stm32mp1***: builds a stm32mp1 platform
- ***DTB_FILE_NAME=<fdt file name>.dtb***: this must be defined to build the proper target and include the correct DTB file into the final file
- ***AARCH32_SP=<monitor>***: sp_min or optee
- ***STM32MP_xxx=1*** is used to choose the boot device, the “xxx” can be EMMC, SDMMC, RAW_NAND, SPI_NAND or SPI_NOR.
- Or a programming interface (you cannot use ***AARCH32_SP=optee*** with those flags):
STM32MP_UART_PROGRAMMER=1 or ***STM32MP_USB_PROGRAMMER=1***

TF-A Build Flags

Optional flags:

- `DEBUG=1`: add debug information in all binaries
- `V=1`: print verbose compilation traces

Notes:

- The `DTB_FILE_NAME` flag must be set to select the correct board configuration.
- The device tree file for the target must be located in `fdts` folder (`<board>.dts`)

TF-A Build Commands

- Add the environment flags:

```
PC: $> unset LDFLAGS && unset CFLAGS
```

- Compile 2 TF-A binaries: one for flash programming (USB or UART), one for device boot (SD-card, eMMC, SPI-NOR, SPI-NAND or parallel NAND (through FMC)):

```
PC: $> make ARM_ARCH_MAJOR=7 ARCH=aarch32 PLAT=stm32mp1 AARCH32_SP=sp_min  
DTB_FILE_NAME=<board>.dtb STM32MP_UART_PROGRAMMER=1 STM32MP_USB_PROGRAMMER=1
```

```
PC: $> make ARM_ARCH_MAJOR=7 ARCH=aarch32 PLAT=stm32mp1 AARCH32_SP=sp_min  
DTB_FILE_NAME=<board>.dtb STM32MP_EMMC=1 STM32MP_SDMMC=1 STM32MP_RAW_NAND=1  
STM32MP_SPI_NAND=1 STM32MP_SPI_NOR=1
```

- Have a look at:

```
PC: $> alias maketfa
```



life.augmented

Update software on board

- Update via SDCARD

PC: \$> dd if=<tf-a file> of=/dev/<device partition> bs=1M conv=fdatasync

- Update via USB mass storage on U-Boot

- Reboot the board then stop at U-Boot command line

Board: \$> help ums

Board: \$> ums 0 mmc 0 (mmc 0 for SD card, mmc 1 for eMMC, usb0 for USB device)

- Use dd command to write the image

PC: \$> dd if=<tf-a file> of=/dev/sdx1 bs=1M conv=fdatasync

- Update your boot device via STM32CubeProgrammer

Modifying TSV files or using STM32_Programmer_CLI

Clock device tree configuration

RCC configuration is done by the first stage bootloader:

- TF-A for the Trusted boot chain
- U-Boot SPL DDT interactive mode for the DDR tuning tool

The configuration is performed using the device tree mechanism that provides a hardware description of the RCC peripheral.

The clock tree is only used in the device tree of boot chain FSBL, even if the clock tree information is also present in the U-Boot device tree, it is not used during boot by this SSBL.

DT bindings documentation:

- TF-A: *tf-a/docs/devicetree/bindings/clock/st,stm32mp1-rcc.txt*
- U-Boot SPL DDT interactive mode : *doc/device-tree-bindings/clock/st,stm32mp1.txt*

Clock device tree configuration

STM32 level DT configuration:

The clock nodes are located in *stm32mp151.dtsi*, open it to find below nodes definition:

- fixed-clock: clk-lsi, clk-lse, clk-his, clk-hse, clk-csi
- RCC node

```
/ {
...
    clocks {
        clk_hse: clk-hse {
            #clock-cells = <0>;
            compatible = "fixed-clock";
            clock-frequency = <24000000>;
        };
    };
...
    soc {
...
        rcc: rcc@50000000 {
            compatible = "st,stm32mp1-rcc", "syscon";
            reg = <0x50000000 0x1000>;
            #clock-cells = <1>;
            #reset-cells = <1>;
            interrupts = <GIC_SPI 5 IRQ_TYPE_LEVEL_HIGH>;
        };
    };
...
};
```

Clock device tree configuration

Board level DT configuration:

The clock tree is also based on five fixed clocks. They are used to define the state of associated STM32MP1 oscillators.

For external oscillator HSE and LSE, there are four fields are supported:

- "st,bypass" configures the external analog clock source (set HSEBYP, LSEBYP),
- "st,digbypass" configures the external digital clock source (set DIGBYP and HSEBYP, LSEBYP),
- "st,css" activates the clock security system (HSECSSON, LSECSSON),
- "st,drive" (LSE only) contains the value of the drive for the oscillator (see LSEDRV_ defined in the file *stm32mp1-clksrc.h*).

HSI clock, the internally fixed to 64MHZ for STM32mp15 devices, is the clock after HSIDIV divider, the frequency is link to the HSIDIV value.

Clock device tree configuration

Clock node example with:

- all oscillators switched on (HSE, HSI, LSE, LSI, CSI)
- HSI at 64MHZ (HSIDIV = 1/1)
- HSE using a digital external clock at 24MHz
- LSE using an external crystal at 32.768kHz (the typical frequency)

```
/ {  
    clocks {  
        clk_hse: clk-hse {  
            #clock-cells = <0>;  
            compatible = "fixed-clock";  
            clock-frequency = <24000000>;  
            st,digbypass;  
        };  
  
        clk_hsi: clk-hsi {  
            #clock-cells = <0>;  
            compatible = "fixed-clock";  
            clock-frequency = <64000000>;  
        };  
        .....  
  
        clk_csi: clk-csi {  
            #clock-cells = <0>;  
            compatible = "fixed-clock";  
            clock-frequency = <4000000>;  
        };  
    };  
};
```

Clock device tree configuration

The bootloader uses other properties for RCC node ("st,stm32mp1-rcc" compatible):

- secure-status: related to TZEN bit configuration in RCC security register that allows to restrict RCC and PWR registers write access
- st,clksrc: clock source configuration array
- st,clkdiv: clock divider configuration array
- st,pll: specific PLL configuration
- st,pkcs: peripheral kernel clock distribution configuration array.
- All the available clocks are defined as preprocessor macros in *stm32mp1-clks.h* and can be used in device tree sources.

ETZPC device tree configuration

The Extended TrustZone Protection Controller (ETZPC) is used to :

- Configure the TrustZone security for Securable IPs
 - Peripherals security mode can be:
 - Secure: read and write access allowed only to secure world
 - Write-secure: Write access allowed only to secure world, read allowed to any
 - Non-secure: read and write access allowed to any
- Configure the SYSRAM and ROM secure regions size
 - The secure region is defined in multiple of 4KB and at the bottom address
- Configure the MCU Isolation domain with a set of isolable IPs allocated to this MCU domain

ETZPC device tree configuration

DT configuration (STM32 level):

The ETZPC node is in the stm32mp151.dtsi file. It describes the hardware register address, clock and status like:

```
etzpc: etzpc@5C007000 {  
    compatible = "st,stm32-etzpc";  
    reg = <0x5C007000 0x400>;  
    clocks = <&rcc TZPC>;  
    status = "disabled";  
    secure-status = "okay";  
};
```

Note: This device tree part is related to STM32 microprocessors. It must be kept as is, without being modified by the end-user

ETZPC device tree configuration

DT configuration (board level):

The ETZPC node in the board dedicated device tree file is used to configure the status of securable peripherals.

```
&etzpc {  
    st,decprot = <  
        DECPROT(STM32MP1_ETZPC_USART1_ID, DECPROT_NS_RW, DECPROT_UNLOCK)  
        DECPROT(STM32MP1_ETZPC_SPI6_ID, DECPROT_NS_RW, DECPROT_UNLOCK)  
        DECPROT(STM32MP1_ETZPC_I2C4_ID, DECPROT_NS_RW, DECPROT_UNLOCK)  
        DECPROT(STM32MP1_ETZPC_I2C6_ID, DECPROT_NS_RW, DECPROT_UNLOCK)  
        DECPROT(STM32MP1_ETZPC_RNG1_ID, DECPROT_NS_RW, DECPROT_UNLOCK)  
        DECPROT(STM32MP1_ETZPC_HASH1_ID, DECPROT_NS_RW, DECPROT_UNLOCK)  
        DECPROT(STM32MP1_ETZPC_Cryp1_ID, DECPROT_NS_RW, DECPROT_UNLOCK)  
    >;  
};  
  
&etzpc {  
    st,decprot = <  
        DECPROT(STM32MP1_ETZPC_RNG2_ID, DECPROT_MCU_ISOLATION, DECPROT_UNLOCK)  
    >;  
};
```

- TF-A version number

NOTICE: BL2: v2.0(debug):<tag>

- v2.0 is the TF-A version used.
- (debug) : Build mode enable
- <tag> : Git reference resulting from the **git describe** command at build time

- Debug with traces

- Please try to modify the LOG_LEVEL value defined in Makefile, then compare the console log printing.
- Change some VERBOSE settings to INFO.

For example: Change the VERBOSE in function “void set_gpio” to INFO.

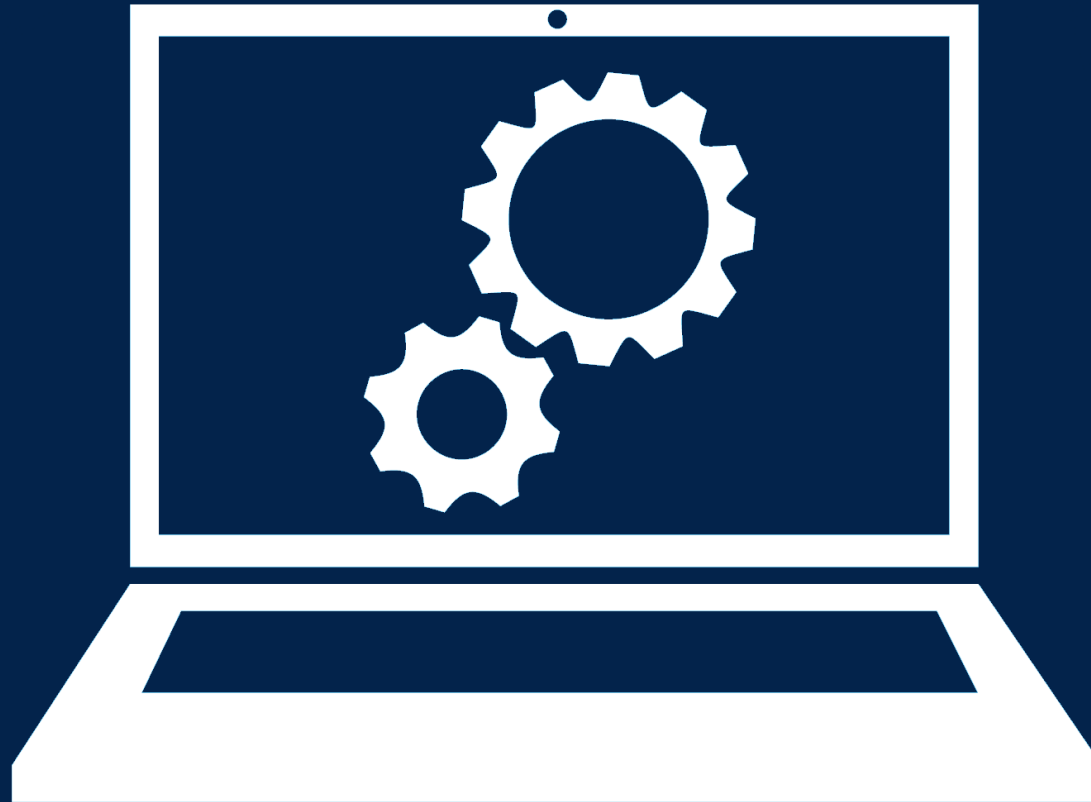
VERBOSE("GPIO %u mode set to 0x%x\n", bank, mmio_read_32(base + GPIO_MODE_OFFSET));



INFO("GPIO %u mode set to 0x%x\n", bank, mmio_read_32(base + GPIO_MODE_OFFSET));

- Setup SDK
 - PC: `$> alias setdevenv`
 - PC: `$> setdevenv`
- Compilation
 - PC: `$> maketfa`
- Update software on board, reset board and stop at U-Boot command line
 - Board: `$> ums 0 mmc 0`
 - PC: `$> sudo dd if=tf-a-stm32mp157c-dk2.stm32 of=/dev/sdx1 bs=1M conv=fdatasync`
- Check the logs
 - Run “ctrl + c” on board side
 - Board: `$> reset`

U-Boot Development



U-Boot Configuration

- defconfig files (link to the boot chain)
configs/stm32mp15_basic_defconfig
configs/stm32mp15_trusted_defconfig
- config file
include/configs/stm32mp1.h
- Drivers
drivers/*/stm32*
- Generic STMicroelectronics board for STM32MP1 Series
board/st/stm32mp1

U-Boot compilation

- Prerequisites (Setup SDK, U-Boot is installed)

- Compilation

- Choose defconfig

PC: `$> make stm32mp15_trusted_defconfig`

If the default config file can not meet your requirement, using `make menuconfig` to change it.

- Compilation for one target

PC: `$> make DEVICE_TREE=stm32mp157x-xxx all`

The supported variables are:

DEVICE_TREE: select in arch/arm/dts the device tree that is used

KBUILD_OUTPUT: change the destination directory for the build

EXT_DTB: select external device tree

Customize boot command

Please try to customize boot command by modifying include/configs/stm32mp1.h

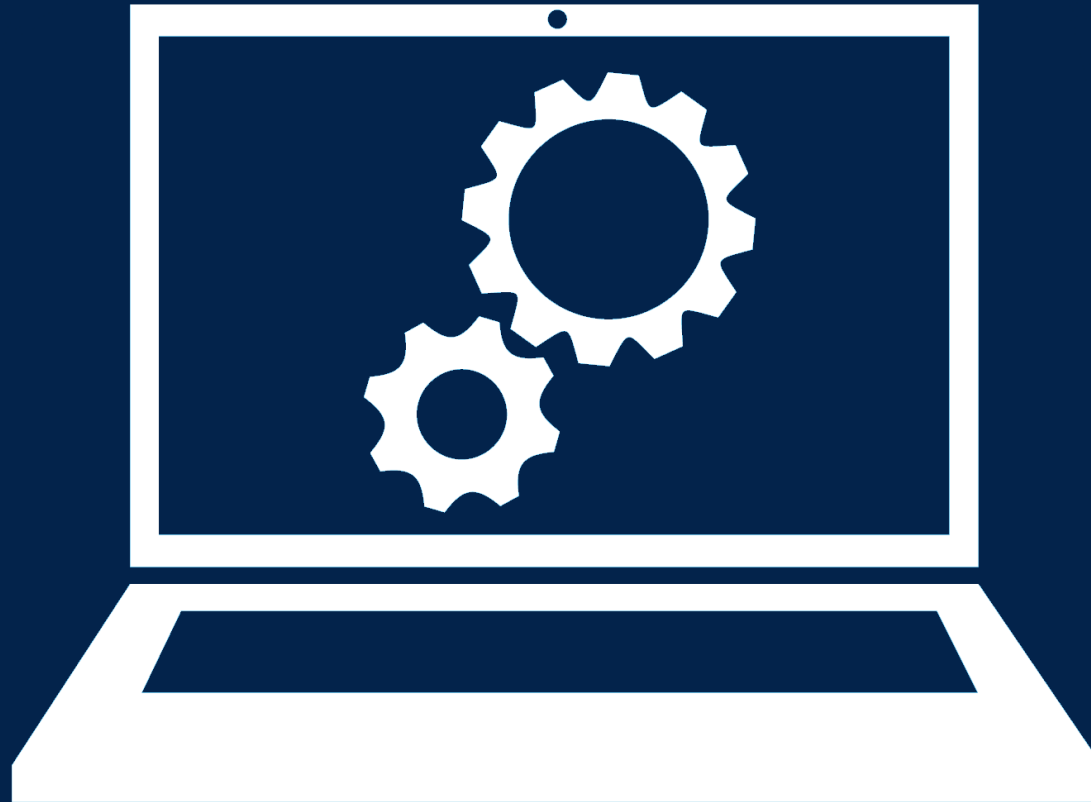
```
--- a/include/configs/stm32mp1.h
+++ b/include/configs/stm32mp1.h
@ -230,12 +230,22 @@ */ #define CONFIG_EXTRA_ENV_SETTINGS \
    "bootdelay=1\0" \
+    "console=ttySTM0,115200\0" \
    "kernel_addr_r=0xc2000000\0" \
    "ramdisk_addr_r=0xc4400000\0" \
+    "loadimage=load mmc 0:4 ${kernel_addr_r} ulimage\0" \
+    "loadfdt=load mmc 0:4 ${fdt_addr_r} ${fdtfile}\0" \
+    "mmcargs=setenv bootargs root=/dev/mmcblk0p6 rw rootfstype=ext4 rootwait\0" \
+    "mmc_boot_test=" \
+    "echo Booting from mmc test.....; " \
+    "run mmcargs; " \
+    "run loadimage; " \
+    "run loadfdt; " \
+    "bootm ${kernel_addr_r} - ${fdt_addr_r}; \0" \
    "altbootcmd=run bootcmd\0" \
```

Customize boot command

- Compile then update to board
 - PC: `$> make DEVICE_TREE=stm32mp157c-dk2 all`
 - PC: `$> cp u-boot.stm32 ../../starter/stm32mp1-openstlinux-5-4-dunfell-mp1-20-11-12/images/stm32mp1/bootloader`
 - Modify TSV file then use CubeProg. to update to board
- Reset the board and stop at U-Boot command line
Board: `$> run mmc_boot_test`
- Set this customize boot command as the autoboot command
 - Redefine CONFIG_BOOTCOMMAND in .config file as following, then compile and update to board to check if the board can autoboot following the customized command:

```
CONFIG_BOOTCOMMAND="run mmc_boot_test"
```

Linux Kernel Development



Linux Kernel compilation

- Configure the Linux kernel
 - For first time compilation
 - PC: `$> cd <directory to kernel source code>`
 - PC: `$> mkdir -p ../build`
 - PC: `$> make ARCH=arm O="$PWD/../build" multi_v7_defconfig fragment*.config`
 - PC: `$> for f in `ls -1 ../fragment*.config`; do scripts/kconfig/merge_config.sh -m -r -O $PWD/../build $PWD/../build/.config $f; done`
 - PC: `$> yes " | make ARCH=arm oldconfig`

For this training, we use the alias command:

- PC: `$> cd /work/developer/linux`
- PC: `$> makekernelconfig`

Linux Kernel compilation

- Configure based on .config
 - Use the menuconfig to add the options, for example, add the debug inside the DMA driver
 - [*] DMA Engine support ->
 - [*] DMA Engine debugging
 - [*] DMA Engine verbose debugging (NEW)

Linux Kernel compilation

- Cross-compile the Linux kernel
 - PC: `$> make ARCH=arm ulmage vmlinux dtbs LOADADDR=0xC2000040 O="$PWD/../build"`
 - PC: `$> make ARCH=arm modules O="$PWD/../build"`
 - PC: `$> make ARCH=arm INSTALL_MOD_PATH="$PWD/install_artifact" modules_install`
 - PC: `$> mkdir -p $PWD/install_artifact/boot/`
 - PC: `$> cp $PWD/arch/arm/boot/umage $PWD/install_artifact/boot/`
 - PC: `$> cp $PWD/arch/arm/boot/dts/st*.dtb $PWD/install_artifact/boot/`

For this training, we just use below alias commands:

- `PC: $> makekernel`
- `PC: $> makekernelmodule`
- `PC: $> makekernelinstall`

Linux Kernel compilation

- Load the generated binary and boot the board
 - The partitions related to the Linux kernel are:
 - the *bootfs* partition that contains the Linux kernel U-Boot image (*ulmage*) and device tree
 - the *rootfs* partition that contains the Linux kernel modules
 - Update via network
 - PC: `$> scp -r $PWD/install_artifact/boot/* root@<ip of board>:/boot/`
 - Update via SDcard
 - PC: `$> cp -r $PWD/install_artifact/boot/* /media/$USER/bootfs/`

SPI loopback test case

- SPI device tree configuration
 - Activate the SPI controller by setting its status to *okay*.
 - Add a spidev child node .
 - Enable spidev by adding a compatible *spidev*.
 - Add a **reg** property, required for the SPI framework but not meaningful in this case since chip select is not defined and loopback is used.
 - Configure the bus speed for SPI communications by setting the **spi-max-frequency** property.

```
&spi4 {  
    status = "okay";  
  
    spidev@0{  
        compatible = "spidev";  
        reg = <0>;  
        spi-max-frequency = <4000000>;  
    };  
};
```

SPI loopback test case

- Kernel configuration, enable *User mode SPI device driver support*

- PC: \$> makekernelconfig

```
[x] Device Drivers
  [x] SPI support
    *** SPI Master Controller Drivers ***
    [x] STMicroelectronics STM32 SPI controller
    *** SPI Protocol Masters ***
    [x] User mode SPI device driver support
```

- Compile and update to board, then reboot:

- PC: \$> makekernel
- PC: \$> scp build/arch/arm/boot/ulmage root@192.168.7.1:/boot
- PC: \$> scp build/arch/arm/boot/stm32mp157c-dk2.dtb root@192.168.7.1:/boot

SPI loopback test case

- Test tool: `spidev_test`

`spidev_test`, available within the Linux kernel, is a test tool enabling to perform tests via the `spidev` interface.

Source code can be found under `tools/spi` directory: `tools/spi/spidev_test.c`

Build and installation:

```
PC: $> cd /work/developer/linux/tools/spi/
```

```
PC: $> make all
```

```
PC: $> scp spidev_test root@192.168.7.1:/usr/sbin
```

SPI loopback test case

- Checking on board

- Hardware preparation:

Short-circuit the SPI bus MISO and MOSI lines to create a loopback enables the bus to receive the same data it is sending. On the STM32MP157X-DKX discovery board, MOSI and MISO signals are accessible via the D12 and D11 pins of connector CN13

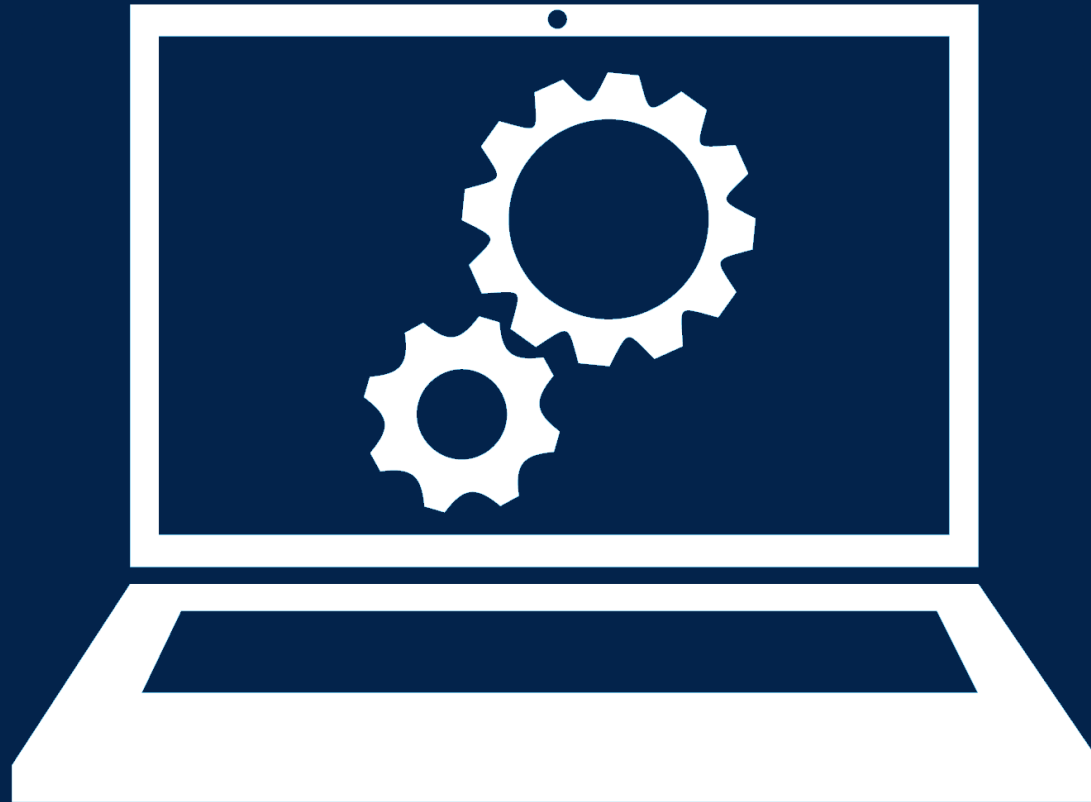
- Check if the spi device has been created

Board: `$> ls /dev/spidev0.0`

- Example of 32-byte transfer in Full-duplex with loopback

Board: `$> spidev_test -D /dev/spidev0.0 -v`

STM32CubeMX & FDCAN Demo



What are you going to do?

On dk2 board, there are two FDCAN interfaces can be used, now, you are required to make these two interface can communicate with each other.

What you have?

Now, you have two CAN transceiver modules, some wires for connecting.

How to do?

- Use CubeMX to complete FDCAN configuration
- Generate the device tree
- Copy the device tree to the kernel source code then do some configuration
- Compile kernel, update ulmage and dtb on board

FDCAN1_RX	PA11
FDCAN1_TX	PA12
FDCAN2_RX	PB12
FDCAN2_TX	PB13



Check the results on board

Device Tree Configuration

- Generation device tree using STM32CubeMX

The screenshot displays the STM32CubeMX software interface. At the top, there is a menu bar with 'File', 'Window', and 'Help' options. Below the menu bar, a dark blue banner contains the 'Home' button and several social media icons (Facebook, YouTube, Twitter, GitHub, and the ST logo). The main content area is divided into three panels:

- Existing Projects:** This panel lists recent opened projects with their names and last modified dates. The projects listed are:
 - FDCAN_TEST.ioc (Last modified date: 25/02/2021 10:59:37)
 - training.ioc (Last modified date: 24/02/2021 16:08:15)
 - training-dk2.ioc (Last modified date: 24/02/2021 16:13:30)
 - emmc_test.ioc (Last modified date: 10/02/2021 12:06:24)
 - STM32MP157-DK1.ioc (Last modified date: 17/11/2020 21:17:11)Below this list is a section for 'Other Projects' with a folder icon.
- New Project:** This panel provides options for starting a new project. It includes a section titled 'I need to :' with three main options:
 - 'Start My project from MCU' with a button labeled 'ACCESS TO MCU SELECTOR'.
 - 'Start My project from ST Board' with a button labeled 'ACCESS TO BOARD SELECTOR' (this button is highlighted with a red circle).
 - 'Start My project from Example' with a button labeled 'ACCESS TO EXAMPLE SELECTOR'.
- Manage software installations:** This panel allows users to manage software installations. It includes a section for checking for updates with a button labeled 'CHECK FOR UPDATES', and a section for installing or removing embedded software packages with a button labeled 'INSTALL / REMOVE'.

At the bottom of the interface, there is a section for 'Build your certified safety system with STM32 and STM8', featuring logos for SIL Ready, ASIL Ready, ClassB Ready, and the ST logo. Below this section are links for 'About STM32' and 'External Tools'.

Device Tree Configuration

Tips:
Initialize all
peripherals with
their default
mode

MX New Project from a Board

MCU/MPU Selector Board Selector Example Selector Cross Selector

Board Filters

Commercial Part Number: **STM32MP157C-DK2** (1)

Vendor: >

Type: >

MCU/MPU Series: >

Other: >

Peripheral: >

Features Large Picture Docs & Resources Datasheet Buy **Start Project** (3)

STM32MP1 Series

STM32MP157C-DK2 **STMicroelectronics STM32MP157C-DK2 Discovery Kit Board Support and Examples**

ACTIVE Active
Product is in mass production

Part Number : STM32MP157C-DK2
Commercial Part Number : STM32MP157C-DK2

Unit Price (US\$) : **99.0**

Mounted Device : [STM32MP157CACx](#)

The STM32MP157A-DK1 and STM32MP157C-DK2 Discovery kits leverage the capabilities of STM32MP1 Series microprocessors to allow users easily develop applications using STM32 MPU OpenSTLinux Distribution software for the main processor and STM32CubeMP1 software for the co-processor.

Boards List: 1 item

	Overview	Commercial Part No	Type	Marketing Status	Unit Price (US\$)	Mounted Device
☆		STM32MP157C-DK2	Discovery Kit	Active (2)	99.0	STM32MP157CACx

Device Tree Configuration

STM32CubeMX Untitled*: STM32MP157CACx STM32MP157C-DK2

File Window Help

Home > STM32MP157CACx - STM32MP157C-DK2 > Untitled - Project Manager > GENERATE CODE

Pinout & Configuration Clock Configuration **Project Manager** Tools

Project

Project Settings

Project Name **fdcan**

Project Location C:\Users\cindy.ge\Downloads\test\ Browse

Application Structure Advanced ☐ Do not generate the m...

Toolchain Folder Location C:\Users\cindy.ge\Downloads\test\fdcan\

Code Generator

Toolchain / IDE **STM32CubeIDE** ☒ Generate Under ...

Advanced Settings

Linker Settings

Minimum Heap Size 0x200

Minimum Stack Size 0x400

MCUs Selection Output DDR Interactive logs

Series	Lines	Mcu	Package	Required Peripherals
STM32MP1	STM32MP157	STM32MP157CACx	TFBGA361	None

Device Tree Configuration

STM32CubeMX Untitled*: STM32MP157CACx - STM32MP157C-DK2

File Window Help

Home > STM32MP157CACx - STM32MP157C-DK2 > Untitled - Pinout & Configuration > GENERATE CODE

Pinout & Configuration

Categories: A-Z

Connectivity

- ETH1
- FDCAN1**
- FMC
- I2C1
- I2C2
- I2C3
- I2C4
- I2C5
- I2C6
- QUADSPI
- SDMMC1
- SDMMC2
- SDMMC3
- SPI1
- SPI2
- SPI3
- SPI4
- SPI5
- SPI6
- UART4
- UART5
- UART7
- UART8
- USART1
- USART2
- USART3
- USART6
- USBH_HS1

FDCAN1 Mode and Configuration

Mode

Boot time: Boot ROM Boot loader Cortex-A7 secure (OP-TEE) **A7NS (Linux)** Cortex-M4 (Stm32Cube)

☒ Activated

Configuration

Reset Configuration

Parameter Settings User Constants GIC Settings **GPIO Settings**

Search Signals Search (Ctrl+F) ☐ Show only Modified

Pin Name	Signal on Pin	GPIO mode	GPIO Pull-up/	Maximum outp.	User Label	Modifi
PA11	FDCAN1_RX	Alternate funct...	No pull-up and...	n/a		☐
PA12	FDCAN1_TX	Alternate Func...	No pull-up and...	Low		☐

TFBGA361 (Top view)

MCUs Selection Output DDR Interactive logs

Series	Lines	Mcu	Package	Required Peripherals
STM32MP1	STM32MP157	STM32MP157CACx	TFBGA361	None

Device Tree Configuration

The screenshot shows the STM32CubeMX interface for configuring the FDCAN2 peripheral. The 'Pinout & Configuration' tab is active, showing the 'FDCAN2 Mode and Configuration' section. The 'Mode' section includes a table for runtime contexts, where 'A7NS (Linux)' is selected. The 'Configuration' section shows the 'GPIO Settings' tab, with a table listing pin configurations for PB12 and PB13. The 'Pinout view' on the right shows the physical pin layout of the TFBGA361 package.

1 FDCAN2 selected in Connectivity

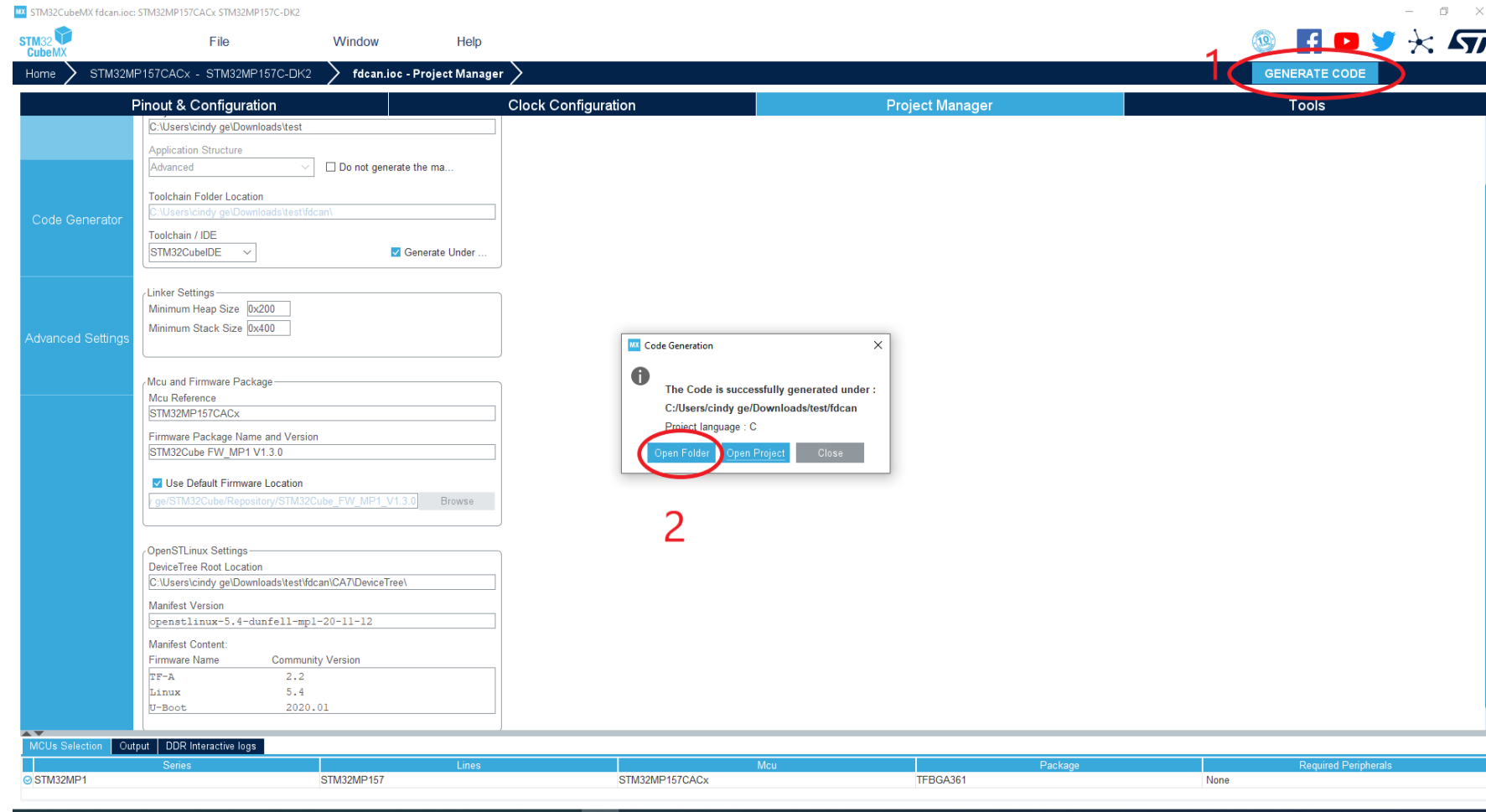
2 'Activated' checkbox

3 A7NS (Linux) runtime context

Pin Name	Signal on Pin	GPIO mode	GPIO Pull-up/	Maximum outp.	User Label	Modifi
PB12	FDCAN2_RX	Alternate funct...	No pull-up and...	n/a		☐
PB13	FDCAN2_TX	Alternate Func...	No pull-up and...	Low		☐

MCUs Selection	Output	DDR Interactive logs
STM32MP1	Series	STM32MP157
	Lines	STM32MP157CACx
	Mcu	TFBGA361
	Package	None
	Required Peripherals	

Device Tree Configuration



- Configure device tree

- Copy kernel device tree *stm32mp157c-fdcan-mx.dts* to */work/developer/linux/arch/arm/boot/dts*
- Modify dts Makefile to compile *stm32mp157c-fdcan-mx.dtb*

```
diff --git a/arch/arm/boot/dts/Makefile b/arch/arm/boot/dts/Makefile
index 1c5d5b0278b2..1ac6cf256ee9 100644
```

```
--- a/arch/arm/boot/dts/Makefile
```

```
+++ b/arch/arm/boot/dts/Makefile
```

```
@@ -1036,6 +1036,7
```

```
@@ dtb-$(CONFIG_ARCH_STM32) += \
```

```
    stm32mp157c-ev1.dtb \
```

```
    stm32mp157d-ev1.dtb \
```

```
    stm32mp157f-ev1.dtb \
```

```
+    stm32mp157c-fdcan-mx.dtb \
    stm32mp157c-ev1-a7-examples.dtb \
    stm32mp157c-ev1-m4-examples.dtb \
```

Kernel Configuration

- Modify kernel FDCAN driver

/work/developer/linux/driver/net/can/m_can/m_can_platform.c

```
diff --git a/drivers/net/can/m_can/m_can_platform.c b/drivers/net/can/m_can/m_can_platform.c
index 38ea5e600fb8..cb8581cb225b 100644
```

```
--- a/drivers/net/can/m_can/m_can_platform.c
```

```
+++ b/drivers/net/can/m_can/m_can_platform.c
```

```
@@ -141,10 +141,10 @@
```

```
static int m_can_plat_remove(struct platform_device *pdev)
```

```
static int __maybe_unused m_can_runtime_suspend(struct device *dev)
```

```
{
```

```
+    pr_info("semico %s, %s, %d\n", __FILE__, __func__, __LINE__);
```

```
    struct net_device *ndev = dev_get_drvdata(dev);
```

```
    struct m_can_classdev *mcan_class = netdev_priv(ndev);
```

```
-
```

```
    m_can_class_suspend(dev);
```

```
    clk_disable_unprepare(mcan_class->cclk);
```

```
    clk_disable_unprepare(mcan_class->hclk);
```

Compilation and Deploy

- *Compile and update to board*

PC: \$> makekernel

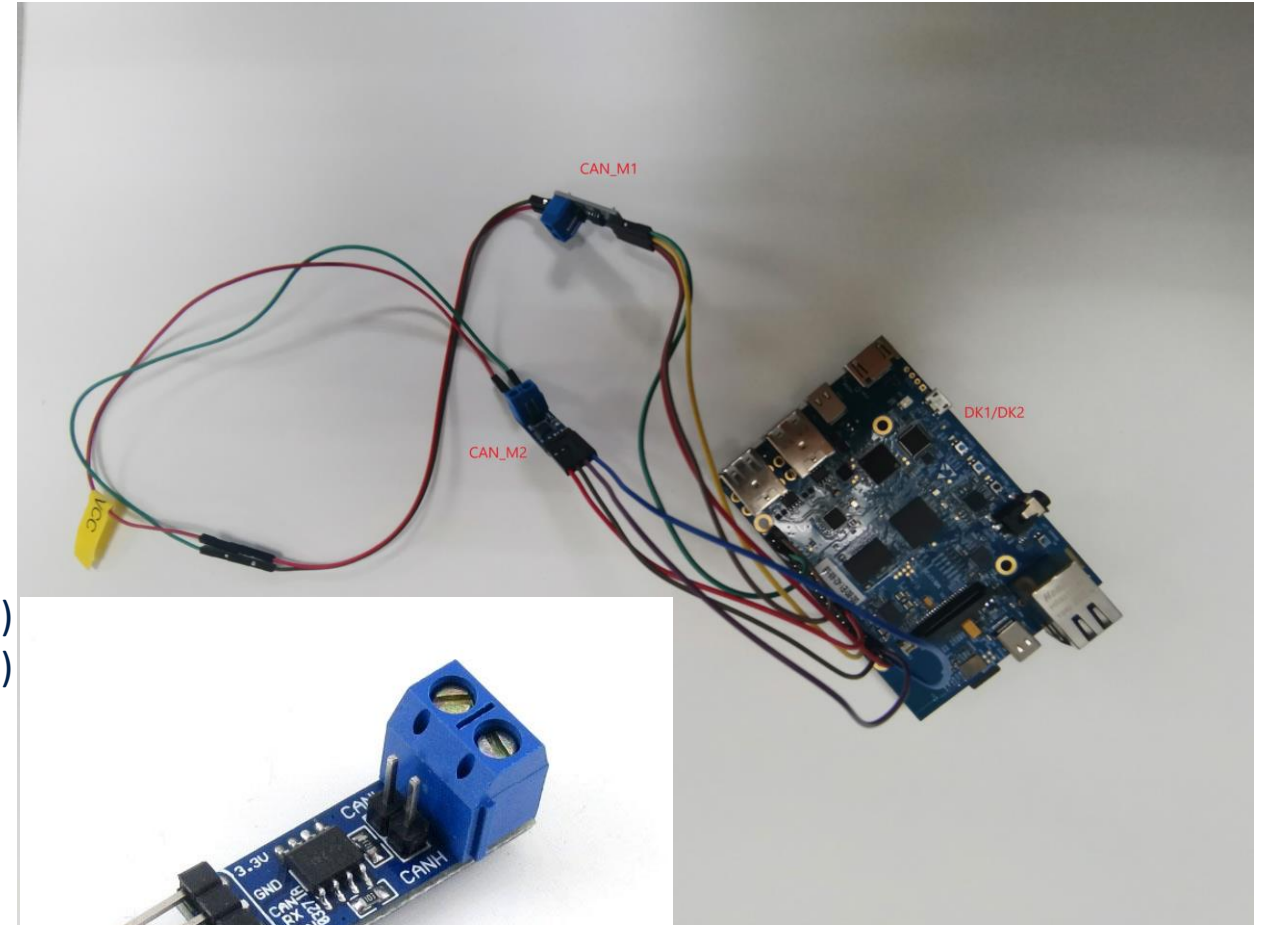
PC: \$> scp build/arch/arm/boot/uImage root@192.168.7.1:/boot

PC: \$> scp build/arch/arm/boot/ stm32mp157c-fdcan-mx.dtb root@192.168.7.1:/boot/ stm32mp157c-dk2.dtb

Hardware Preparation

- Hardware connection

CAN_M1: CANL	←	→	CAN_M2: CANL
CAN_M1: CANH	←	→	CAN_M2: CANH
CAN_M1: 3.3V	←	→	DK2: 3V3 (CN2 pin1)
CAN_M1: GND	←	→	DK2: GND (CN2 pin39)
CAN_M1: CAN_RX	←	→	DK2: EXP_GPIO3 (CN2 pin5)
CAN_M1: CAN_TX	←	→	DK2: EXP_GPIO2 (CN2 pin3)
CAN_M2: 3.3V	←	→	DK2: 3V3 (CN2 pin17)
CAN_M2: GND	←	→	DK2: GND (CN2 pin6)
CAN_M2: CAN_RX	←	→	DK2: EXP_GPIO15 (CN2 pin10)
CAN_M2: CAN_TX	←	→	DK2: EXP_GPIO16 (CN2 pin36)



FDCAN Demo Show

- Power on to test the demo

- Board: `$> dmesg | grep m_can`

- Board : `$> ip link set can0 type can bitrate 100000 dbitrte 200000 fd on`

- Board : `$> ip link set can1 type can bitrate 100000 dbitrte 200000 fd on`

- Board : `$> ip link set can0 up`

- Board : `$> ip link set can1 up`

- Board : `$> candump -L can1 &`

- Board : `$> cansend can0 123#1122334455667788`

- Board : `$> cansend can0 123#1122334455667788`

```
root@stm32mp1:~# candump -L can1 &
root@stm32mp1:~# cansend can0 123#1122334455667788
root@stm32mp1:~# (1581090731.440684) can1 123#1122334455667788
root@stm32mp1:~# cansend can0 123#1122334455667788
(1581090732.804869) can1 123#1122334455667788
root@stm32mp1:~# cansend can0 123#1122334455667788
(1581090733.531025) can1 123#1122334455667788
root@stm32mp1:~# cansend can0 123#1122334455667788
(1581090734.194177) can1 123#1122334455667788
```

Questions

- Add log information “ST DFAE training hands on” in TF-A source code, compile and update TF-A image to board, then the new log can be printed.(10 pts)
Note: Pay attention to the current LOG_LEVEL setting.
- Change the “bootdelay” value to 3 in U-Boot, then update U-Boot image to board. (10 pts)
- We have done the two FDCAN interface pinout configuration by STM32CubeMX, please directly modify the device tree *stm32mp157c-dk2.dts* and *stm32mp15-pinctrl.dtsi* to complete the pinout configuration, then generate a patch file named *fdcan.patch*.(10 pts)

Note: The patch generation command:

```
PC: $> git diff > fdcan.patch
```

Thank you