

1 Introduction

This application note will help to configure STM32CubeMX to work with [the OSD32MP15x, the STM32MP1 System-in-Package](#).

STM32CubeMX (commonly called just “CubeMX”) is a STMicroelectronics processor configuration tool with an intuitive Graphical User Interface (GUI) that has helped programmers and embedded designers configure their ST microcontrollers with ease. With the launch of STM32MP15x, the ST CubeMX tool has been extended to support the MP1 microprocessor family as well, specifically the STM32MP15x, which is integrated within the Octavo Systems OSD32MP15x SiP, the STM32MP1 System in Package. As a result, previous ST microcontroller users can easily apply their knowledge and experience with CubeMX to configure the OSD32MP15x. Through a step-by-step process, STM32CubeMX will help generate initialization C code for the ARM® Cortex®-M4 core and partial Linux® Device Trees for ARM® Cortex®-A7 cores. Apart from code generation, CubeMX can also help perform power consumption estimation and DDR testing.

Generating initialization code for the STM32MP1 using CubeMX involves configuring Pinout, Peripherals, and Clocks. An example screenshot of the “Pinout & Configuration” window is shown in Figure 1.

This application note provides pre-configured CubeMX project files in the ***OSD32MP15x_MinimalConfig.zip*** file. The minimal pinout, peripheral and clock configuration for the OSD32MP1 is already completed in this project so that you can quickly get started with the OSD32MP15x specific SiP device. You can use these configuration settings as a starting point to make modifications for your custom design.

In addition, this app note will briefly describe the STM32MP1 CubeMX settings which you should and should not modify for the OSD32MP15x. It will also point you to important documentation and user manuals to assist you. The tutorial assumes that you are already familiar with the OSD32MP15x device and the integrated peripherals and have a basic understanding of CubeMX. If not, please use the links in Section 2 to get more information.

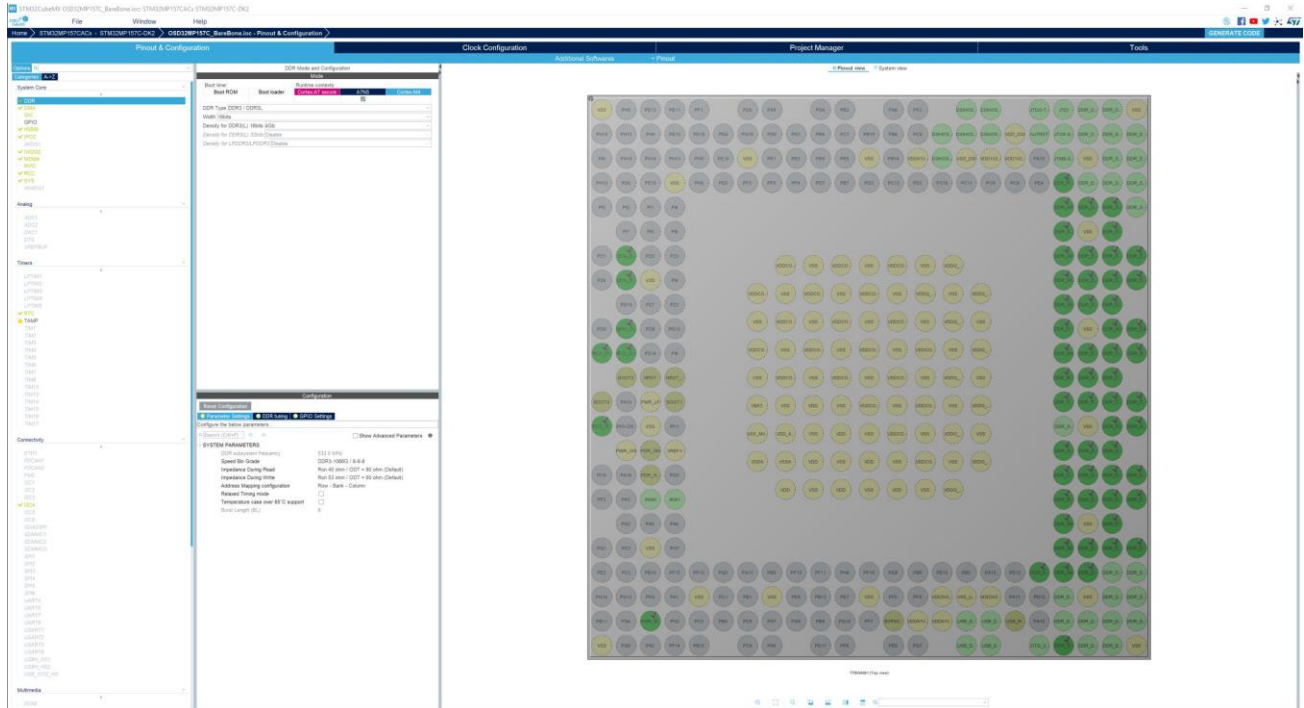


Figure 1 - CubeMX Pinout & Configuration Window

Notice: The information provided within this document is for informational use only. Octavo Systems provides no guarantees or warranty to the information contained.

Table of Contents

1	Introduction.....	1
2	Prerequisites.....	4
3	Minimal CubeMX Project	5
3.1	Project Naming Conventions.....	5
3.2	STM32MP1 CubeMX Configuration for the Octavo Systems OSD32MP15x, the STM32MP1 System in Package (SiP).....	5
3.2.1	Fixed Settings	5
3.2.2	Customizable Settings	6
3.3	CubeMX Project Manager	6
3.4	CubeMX Tools.....	7
4	Generating code using CubeMX.....	8
5	Flashing and upgrading SD card	8
6	Using the U-Boot Device Tree	11
7	Using the Trusted Firmware Device Tree	15
8	Using the Linux Device Tree	18
9	Transition from U-Boot to Linux Kernel	20
10	Resources	21
11	Revision History.....	22

2 Prerequisites

1. Download and install **CubeMX 5.3.0**:
<https://www.st.com/en/development-tools/stm32cubemx.html#overview>
2. Download **OSD32MP15x_MinimalConfig.zip** file:
https://octavosystems.com/files/osd32mp15x_minimalconfig/
3. Unzip the file and open the **<OCTAVO PART NUMBER>_MinimalConfig** project file corresponding to your OSD32MP15x part number from CubeMX:
“File” → “Load Project” → Choose the **<OCTAVO PART NUMBER>_MinimalConfig.ioc** file.
4. Background Information:
 - a) CubeMX User Manual:
https://www.st.com/resource/en/user_manual/dm00104712.pdf
 - b) STM32MP157x datasheet:
<https://www.st.com/resource/en/datasheet/stm32mp157c.pdf>
 - c) STM32MP157x Reference Manual:
https://www.st.com/resource/en/reference_manual/dm00327659.pdf
 - d) OSD32MP15x Datasheet
<https://octavosystems.com/docs/osd32mp15x-datasheet/>
 - e) Pin Mapping between STM32MP15x and OSD32MP15x:
https://octavosystems.com/app_notes/osd32mp15x-pin-mapping-to-stm32mp1/

3 Minimal CubeMX Project

The minimal pinout, peripheral and clock configuration for the OSD32MP157x is provided in the CubeMX projects contained in the ***OSD32MP15x_MinimalConfig.zip*** file to help you quickly get started with OSD32MP15x. The projects were created using CubeMX v5.3.0.

3.1 Project Naming Conventions

The files within the ***OSD32MP15x_MinimalConfig.zip*** file follow the naming convention:

“<OCTAVO PART NUMBER>_MinimalConfig.ioc”

For the remainder of the document, we will refer to these files collectively as “minimal CubeMX Project”. Additionally, when <OCTAVO PART NUMBER> appears in the name of a file, we will just use the abbreviated <DEVICE> to refer to this part of the name.

The ***OSD32MP15x_MinimalConfig.zip*** will only contain CubeMX projects for commercial part numbers. Since the functionality of OSD32MP15x industrial parts are same as that of their commercial counterparts, the CubeMX project and associated device tree files for a commercial part number can be used for the corresponding industrial part number as well.

3.2 STM32MP1 CubeMX Configuration for the Octavo Systems OSD32MP15x, the STM32MP1 System in Package (SiP)

The CubeMX configuration that is already available for OSD32MP157C SiP as part of <OCTAVO PART NUMBER>_MinimalConfig.ioc project can be classified into two groups:

- Fixed Settings
- Customizable Settings

These settings are discussed in the following sections.

3.2.1 Fixed Settings

Since OSD32MP15x SiP integrates several components along with the STM32MP15x processor, the following CubeMX settings correspond to components within the SiP that are fixed and therefore these settings should NOT be changed. The fixed settings are also part number specific. Make sure to use the CubeMX project that corresponds to the exact part number of OSD32MP15x device you are using in your design. The fixed settings in the minimal CubeMX Project are:

- DDR
 - DDR Mode and Configuration settings in “System Core”
- GPIO
 - DDR Pin Configuration settings in “GPIO”
 - I2C4_SCL/I2C4_SDA Pin Configuration settings (enables communication between processor and PMIC)

- UART4 Pin Configuration settings (enables serial debug console)
- PC13 is reserved for PMIC_WAKEUP
- PC14-OSC32_IN, PC15-OSC32_OUT and PH0_OSC_IN pins are reserved for clock circuitry
- RCC
 - RCC Mode and Configuration settings in “System Core”
- RTC
 - RTC Mode and Configuration settings in “Timers”
- Clock Configuration
 - The clock configuration of the processor in the “Clock Configuration” tab
- I2C4
 - I2C4 Mode and Configuration settings in “Connectivity”
- UART4
 - UART4 Mode and Configuration settings in “Connectivity”. UART4 is used as the default serial console interface by the processor.
- SDMMC1
 - SDMMC1 Mode and Configuration settings in “Connectivity”.

3.2.2 Customizable Settings

All the settings that are not part of the Fixed Settings can be customized to suit your design needs. Some commonly customized settings are:

- GPIO – except reserved pins mentioned above
- DMA
- GIC
- IPCC
- NVIC
- I2Cx
- UARTx
- SPIx
- USBx
- BSEC
- ETZPC
- RNG1
- OPENAMP

See the Octavo Systems app note: [Pin Mapping Between OSD32MP15x , the STM32MP1, and Other Integrated Devices](#), to understand how the pins of the discrete STM32MP157x processor are mapped to the OSD32MP157x System in Package (SiP).

3.3 CubeMX Project Manager

The CubeMX Project Manager can generate partial Linux® Device Trees for the ARM® Cortex®-A7 cores. It also can generate initialization C code for the ARM® Cortex®-M4 core for several tool-chains and IDEs. You are free to use any supported tool-chain. However, the M4 initialization C code of the minimal CubeMX Project has only been tested on the *System Workbench for STM32 IDE* (i.e., SW4STM32) with

the *STM32Cube FW_MP1 V1.0.1* firmware package. The partial device tree output files of the minimal CubeMX Project have been tested on the OSD32MP157C running *OpenSTLinux Developer Package v1.0.0*.

To learn more about these tools, see section 4.8 of the [CubeMX manual](#).

Caveat:

CubeMX does not support the configuration/management of all the peripherals through its GUI (as indicated by WARNINGS in the GUI). Hence, it is necessary to manually inspect and modify/add to the device tree once it has been generated by CubeMX.

To learn more about Linux Device Trees, see [OSD335x Lesson 2: Linux Device Tree](#).

3.4 CubeMX Tools

CubeMX features additional developmental tools like the *Power Consumption Calculator* and *DDR Test Suite*. These tools can also help you make your design process easier and do not require any modifications to use with the OSD32MP15x. The CubeMX Tools window is shown in Figure 2.

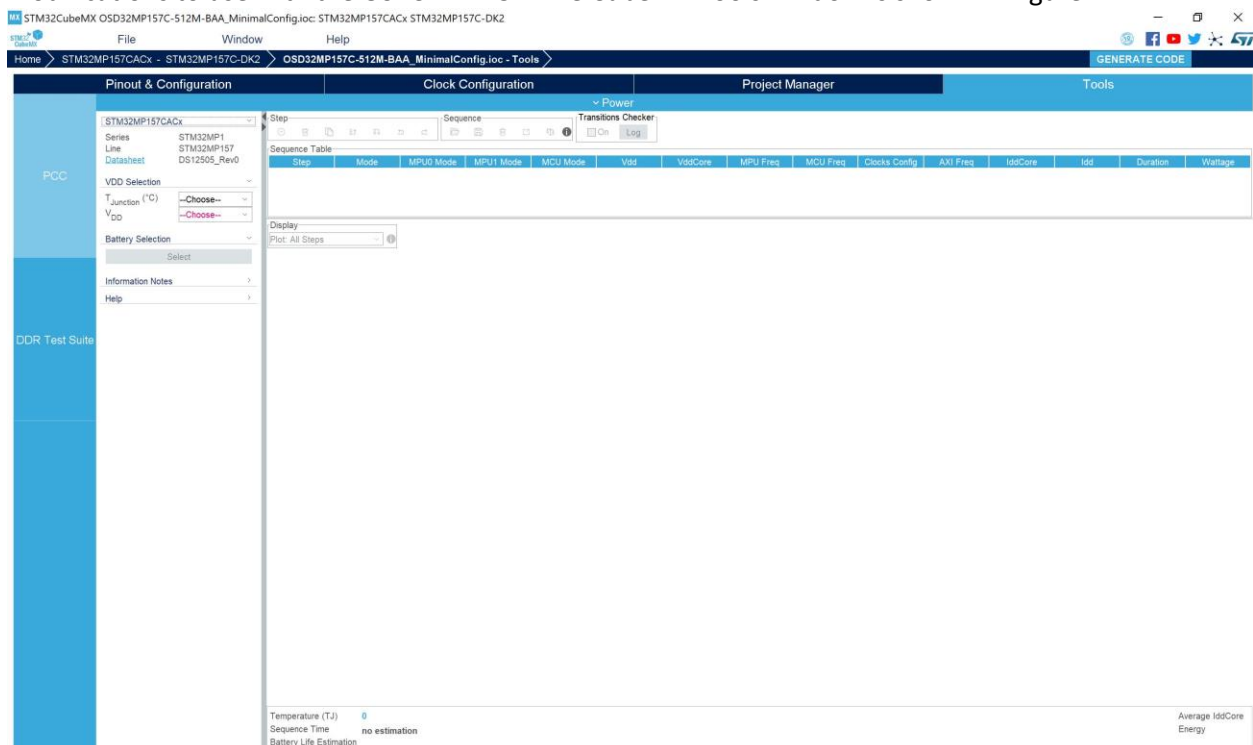


Figure 2 - CubeMX Tools

To learn more about these tools, see Sections 5.1 and 5.2 of the [CubeMX manual](#).

4 Generating code using CubeMX

As mentioned before, CubeMX is capable of generating initialization code for both the Dual Core Cortex-A7 microprocessor cores and M4 microcontroller core. Before proceeding to generate the code, make sure the Pinout & Configuration, Clock Configuration and Project settings match your expectations.

To generate initialization code for A7 core and M4 core:

- Press the **GENERATE CODE** button at the top right corner of the CubeMX GUI.
- The generated device tree files for the A7 cores will be available under: **<CubeMX project directory>/<CubeMX project name>/DeviceTree/<Device name>** directory. In this directory, you should be able to see three other directories **kernel**, **tf-a** and **u-boot** (more information about using these device tree files is given in the next section):
 - **kernel** directory will contain device tree files for the linux kernel.
 - **u-boot** directory will contain device tree files for the secure and non-secure boot chains of u-boot.
 - **tf-a** will contain device trees files for the secure First Stage Boot Loader (FSBL) of trusted boot chain.
- Similar to A7 core, the generated M4 initialization code (i.e., SW4STM32 IDE project) will be available under your CubeMX project directory. To make use of the generated code, first download and install the [System Workbench for STM32 \(also called SW4STM32\) IDE](#). Once installed, open the IDE, go to File > Import > Existing Projects into Workspace > Browse and select the **SW4STM32** directory (which was generated by CubeMX) under your CubeMX project directory > click Finish.
- Once the initialization project code is imported, you can modify it/add to it to suit your design requirements and build/deploy it on the M4 core of your OSD32MP15x using the same IDE. To learn more about SW4STM32 IDE, see Overview and Documentation of SW4STM32 at [OpenSTM32 Community](#) (registration and login required).

Caveat:

CubeMX automatically chooses the CubeMX project file directory as the default directory for generated device tree files and SW4STM32 IDE project files. The root directory for the generated code cannot be manually set.

5 Flashing and upgrading SD card

Before you can use the custom device tree files generated from CubeMX, you need to first flash a SD card with OpenSTLinux Starter Package. The starter package is only intended to demonstrate the capabilities of the DK2 board using demo projects. It does not support custom modifications to Linux kernel. Hence, it is also necessary to upgrade the Starter package to Developer package before proceeding to the next section. To flash and upgrade the SD card, follow the below steps:

Step 1: Flashing the SD Card

To use a custom Device Tree, it is necessary to upgrade OpenSTLinux Starter Package to Developer Package. To do this, please complete the following steps:

- i. Install the **STM32CubeProgrammer** tool on a host Ubuntu computer using the instructions in Section 5.1 of the [ST Starter Package Wiki page](#).
- ii. Prepare the USB serial link on the host Ubuntu computer using the instructions in Section 5.2 of [ST Starter Package Wiki page](#).
- iii. Download the Starter Package image on the host Ubuntu computer and flash your SD card with it using instructions given in Section 6 of [ST Starter Package Wiki page](#).
- iv. Boot the board and check the boot sequence using the instructions given in Section 7 and 8 of the [ST Starter Package Wiki page](#). The UART4 serial console should lead you to a login screen at the end of boot messages as shown in Figure 3 and the DK2 LCD screen should display a demo apps page as shown in Figure 4.

```
[ OK ] Started Netdata, Real-time performance monitoring.
[ OK ] Started Network Name Resolution.
[ OK ] Reached target Network.
        Starting Target Communication Framework agent...
[ OK ] Started IIO Daemon.
        Starting Permit User Sessions...
        Starting Avahi mDNS/DNS-SD Stack...
[ OK ] Reached target Host and Network Name Lookups.
[ OK ] Started Target Communication Framework agent.
[ OK ] Started Permit User Sessions.
[ OK ] Started Serial Getty on ttySTM0.
[ OK ] Started Getty on tty1.
[ OK ] Reached target Login Prompts.
[ OK ] Started Avahi mDNS/DNS-SD Stack.
[ OK ] Reached target Multi-User System.
        Starting Update UTMP about System Runlevel Changes...
[ OK ] Started Update UTMP about System Runlevel Changes.
[ 17.769798] cfg80211: Loading compiled-in X.509 certificates for regulatory database
[ 17.810285] cfg80211: Loaded X.509 cert 'sforshee: 00b28ddf47aef9cea7'

ST OpenSTLinux - Weston - (A Yocto Project Based Distro) 2.6-openstlinux-4.19-thud-mp1-19-02-20 stm32mp1 ttySTM0
stm32mp1 login: root (automatic login)

root@stm32mp1:~# █
```

Figure 3 - OSD32MP15x Starter Pack Login Screen

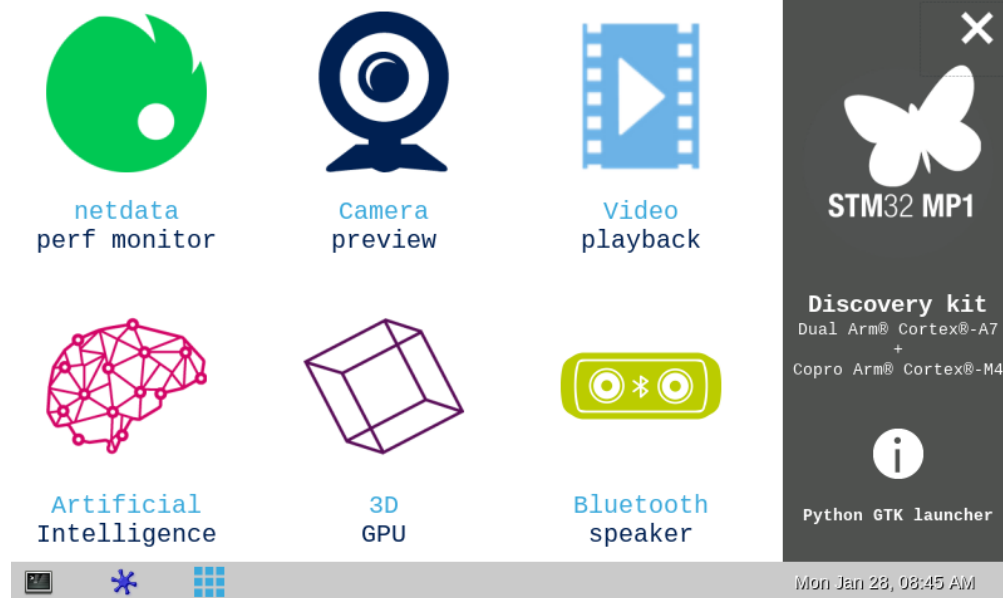


Figure 4 Demo Apps screen on DK2 LCD panel (© STMicroelectronics)

Caveat:

Please follow the directory structure recommended under “[proposition to organize the tools’ directory](#)” to facilitate the steps in this guide. The instructions assume you are using the recommended directory structure. Additionally:

- Make sure the **PATH** variable is set to the install directory of the **STM32CubeProgrammer** as recommended in the instructions. You may get “**STM32_Programmer_CLI command not found**” errors if the path variable is not set properly.
- The BOOT switches on the back of DK2 board should be modified as instructed before and after SD card flashing process.

Step 2: Upgrading to the Developer Package

Upgrading to the Developer Package of the OpenST Linux distribution involves installing an SDK for cross-development on the host Ubuntu computer and then installing the Developer Linux Kernel on the SD card. To do this, please complete the following steps:

- Install the SDK on the host Ubuntu computer using the instructions given in Section 5.1 of the [ST Developer Wiki Page](#).
- Install the Developer Package Linux Kernel on the SD card using the instructions given in Section 5.2 of the [ST Developer Wiki Page](#) (see Caveat below).
- Once the installation of Developer Linux Kernel finishes, reboot the board to see if there are any errors in the boot messages. If not, proceed to the next step.

Caveat:

The Linux Kernel build process discussed under section 5.2 of the [ST Developer Wiki Page](#) proposes two ways to build to kernel:

- Compile and install in a build directory
- Compile and install in the current source code directory

We recommend using the first method (i.e., using a separate build directory) to enable better flexibility while making modifications to the build.

6 Using the U-Boot Device Tree

This section will guide you through the process of building a custom U-Boot that uses the custom U-Boot Device Tree generated from the minimal CubeMX project. The following steps assume that you are using the **STM32MP157C-DK2** development platform from ST. It will be referred to as the “DK2 board” here after. The process of loading a custom U-Boot Device Tree should be similar for your custom design.

The U-Boot device tree generated by CubeMX consists of 3 files (more info on this topic [here](#)):

- A copy of the Linux device tree as the main .dts file - ***stm32mp157c-<DEVICE>_minimalconfig-mx.dts***
- A .dtsi include file with U-Boot specific configuration – ***stm32mp157c-<DEVICE>_minimalconfig-mx-u-boot.dtsi***
- A header file for DDR configuration - ***stm32mp15-mx.h***

The steps to build U-boot using a custom Device Tree are as follows:

Step 1: Download and install U-Boot Developer Package on Host Computer

Before making custom edits to the U-Boot device tree or its source code, we need to first download and install U-Boot Developer Package on Host Ubuntu computer. To do so, follow the steps under section 5.3.1 of [ST Developer Package Wiki page](#).

Step 2: Build and deploy default U-Boot

The U-Boot developer package provided by ST supports three boot chains (more on this topic [here](#)):

- **Basic boot chain** – First Stage Boot Loader (FSBL) is U-Boot SPL (Secondary Program Loader) and Second Stage Boot Loader (SSBL) is U-Boot.
- **Trusted boot chain** – FSBL is Trusted Firmware-A (TF-A) and SSBL is U-Boot. The Starter OpenSTLinux image uses this boot chain by default.
- **OP-TEE** – Secure OS, i.e. a Trusted Execution Environment. This boot chain is beyond the scope of this application note but you can learn more about it [here](#).

It is recommended to test U-Boot in its default form before making modifications to it. Build and deploy U-Boot using the steps given under section 5.3.2 of [ST Developer Package Wiki page](#). In the compilation step, compile for several default targets using **make -f \$PWD/./Makefile.sdk all**.

- Test the basic tool chain by using the FSBL: **u-boot-spl.stm32-stm32mp157c-dk2-basic** and SSBL: **u-boot-stm32mp157c-dk2-basic.img** under **build-basic** directory. The boot log from the UART4 serial console for basic tool chain should look similar to Figure 7.
- Test the trusted boot chain by using FSBL: **tf-a-stm32mp157c-dk2-trusted.stm32** which is provided by TF-A Dev package (see Section 7) and SSBL: **u-boot-stm32mp157c-dk2-trusted.stm32** from build-trusted directory. The boot log for trusted boot chain should look like Figure 9.

Step 3: Modifying the default U-Boot source code

You can validate your toolchain and U-Boot build process by adding custom boot log text to the U-Boot source code using the example steps given under Section 7 of [How to Cross-compile ST Wiki package](#). After modifying, building and deploying U-Boot on your board, the boot log should look similar to Figure 5. Please note that the actual phrase used in the example picture is different from what is suggested in the Wiki page. Feel free to put in any phrase you desire.

```
U-Boot SPL 2018.11-stm32mp-r2 (Jul 09 2019 - 14:18:32 -0500)
Model: STMicroelectronics STM32MP157C-DK2 STM32CubeMX board
RAM: DDR3-DDR3L 16bits 533000Khz
Trying to boot from MMC1

U-Boot 2018.11-stm32mp-r2 (Jul 09 2019 - 14:18:32 -0500)

CPU: STM32MP157CAC Rev.B
Model: STMicroelectronics STM32MP157C-DK2 STM32CubeMX board
U-Boot Basic Mode Testing.....
Board: stm32mp1 in basic mode (st,stm32mp157c-osd32mp157c-512m-baa_minimalconfig-mx)
Board: MB1272 Var2 Rev.C-01
DRAM: 512 MiB
Clocks:
- MPU : 650 MHz
- MCU : 208.878 MHz
- AXI : 266.500 MHz
- PER : 0 MHz
```

Figure 5 - Adding custom boot messages to basic boot chain log

Step 4: Building U-Boot from Custom Device Trees

- Copy the Custom U-Boot Device Tree source files (i.e., **stm32mp157c-<DEVICE>_minimalconfig-mx.dts**, **stm32mp157c-<DEVICE>_minimalconfig-mx-u-boot.dtsi** and **stm32mp15-mx.h**) to the **<U-Boot source code directory>/arch/arm/dts/** directory of the host Ubuntu computer.
- Navigate to U-Boot source directory using the command:
cd <U-Boot source directory>
- Build U-Boot using the commands:

- If running **make** for the first time, use: **make -f \$PWD/./Makefile.sdk all**
 - After first time, use: **make -f \$PWD/./Makefile.sdk all DEVICE_TREE="stm32mp157c-<DEVICE>_minimalconfig-mx"**
- iv. Navigate one step above in the directory hierarchy from **<U-Boot source code directory>** using the command **cd ../** and view the contents using the **ls** command. The U-Boot build files for different boot chains will be available under **build-basic**, **build-trusted** and **build-optee** directories.

Step 5a: Configuring SD card to use custom U-Boot (Basic boot chain)

- i. Navigate to basic tool chain directory:
cd build-basic
- ii. Insert the SD card (pre-flashed with OpenSTLinux Developer image built as part of Section 5) to the host Ubuntu computer.
- iii. Observe the partitions of the SD card by using the command: **ls -l /dev/disk/by-partlabel/**. The part labels should look like Figure 6.

```
lab@lab-machine:~$ ls -l /dev/disk/by-partlabel
total 0
lrwxrwxrwx 1 root root 10 Jul  8 21:30 bootfs -> ../../sdb4
lrwxrwxrwx 1 root root 10 Jul  8 21:30 fsbl1 -> ../../sdb1
lrwxrwxrwx 1 root root 10 Jul  8 21:30 fsbl2 -> ../../sdb2
lrwxrwxrwx 1 root root 10 Jul  8 21:30 rootfs -> ../../sdb6
lrwxrwxrwx 1 root root 10 Jul  8 21:30 ssbl -> ../../sdb3
lrwxrwxrwx 1 root root 10 Jul  8 21:30 userfs -> ../../sdb7
lrwxrwxrwx 1 root root 10 Jul  8 21:30 vendorfs -> ../../sdb5
lab@lab-machine:~$
```

Figure 6 - SD card partition labels

- iv. U-Boot SPL (u-boot-spl.stm32-*) MUST be copied on the dedicated partition named "fsbl1" of the SD card using the command:
dd if=u-boot-spl.stm32-stm32mp157c-<DEVICE>_minimalconfig-mx-basic of=/dev/sdb1 bs=1M conv=fdatasync
- v. U-Boot image (u-boot*.img) MUST be copied on the dedicated partition named "ssbl" of the SD card using the command:
dd if=u-boot-stm32mp157c-<DEVICE>_minimalconfig-mx-basic.img of=/dev/sdb3 bs=1M conv=fdatasync
- vi. Safely eject the SD card from the host computer once the transfer is complete.
- vii. Insert the SD card back into the DK2 board. Power the board. The boot log on UART4 console for basic tool chain should look similar to Figure 7.

```

U-Boot SPL 2018.11-stm32mp-r2 (Jul 07 2019 - 23:58:33 -0500)
Model: STMicroelectronics STM32MP157C-DK2 STM32CubeMX board
RAM: DDR3-DDR3L 16bits 533000Khz
Trying to boot from MMC1

U-Boot 2018.11-stm32mp-r2 (Jul 07 2019 - 23:58:33 -0500)

CPU: STM32MP157CAC Rev.B
Model: STMicroelectronics STM32MP157C-DK2 STM32CubeMX board
Board: stm32mp1 in basic mode (st,stm32mp157c-osd32mp157c-512m-baa_minimalconfig-mx)
Board: MB1272 Var2 Rev.C-01
DRAM: 512 MiB
Clocks:
- MPU : 650 MHz
- MCU : 208.878 MHz
- AXI : 266.500 MHz
- PER : 0 MHz
- DDR : 533 MHz
board_check_usb_power: Can't get adc device(-19)
NAND: 0 MiB
MMC: STM32 SDMMC2: 0
In: serial
Out: serial
Err: serial
Net: No ethernet found.
Boot over mmc0!
Hit any key to stop autoboot: 0

```

Figure 7 - Basic Boot Chain Boot Log

Step 5b: Configuring SD card to use custom U-Boot (Trusted boot chain)

- i. Navigate to trusted (TF-A) tool chain directory:
cd build-trusted
- ii. Insert the SD card (pre-flashed with OpenSTLinux Developer image developed as part of Section 5) to the host Ubuntu computer.
- iii. Observe the partitions of the SD card by using the command: ***ls -l /dev/disk/by-partlabel/***. The part labels should look like Figure 6.
- iv. In this boot chain, the First Stage Boot Loader (FSBL) is provided by TF-A (see Section 7). The U-Boot binary (u-boot*.stm32) acts as Second Stage Boot Loader (SSBL). Copy only the U-Boot binary (u-boot*.stm32) to the dedicated partition named "ssbl" of the SD card using the command:
dd if=u-boot-stm32mp157c-<DEVICE>_minimalconfig-mx-trusted.stm32 of=/dev/sdb3 bs=1M conv=fdatasync
- v. See Section 7, step 5 for instructions to load FSBL.

Caveat:

1. In the case of secure boot, since FSBL uses TF-A device tree and SSBL uses U-Boot device tree, make sure the ***compatible property*** of both these device trees are identical. Otherwise, the boot sequence may not transition from FSBL to SSBL.
2. Irrespective of the boot chain being used (basic or secure), a directory whose name corresponds to the ***compatible property*** of U-Boot should exist in the ***bootfs*** partition of SD card. U-Boot will use boot parameters defined in this directory to boot linux kernel. If U-boot fails to find this directory, the boot sequence will halt and will not transition to linux kernel. See section 9 for more information.

7 Using the Trusted Firmware Device Tree

This section will guide you through the process of building a custom Trusted Firmware A (TF-A) that uses the custom TF-A Device Tree generated from the minimal CubeMX project. The following steps assume that you are using the *STM32MP157C-DK2* development platform from ST. It will be referred to as the “DK2 board” here after. The process of loading a custom TF-A Device Tree should be similar for your custom design.

The TF-A Device Tree generated by CubeMX consists of 2 files:

- A TF-A device tree file - *stm32mp157c-<DEVICE>_minimalconfig-mx.dts*
- A header file for DDR configuration - *stm32mp15-mx.h*

The steps to build TF-A using a custom Device Tree are as follows:

Step 1: Download and install TF-A Developer Package on Host Computer

Before making custom edits to the TF-A device tree or its source code, we need to first download and install TF-A Developer Package on a Host Ubuntu computer. To do so, follow the steps under section 5.4.1 of [ST Developer Package Wiki page](#).

Step 2: Build and deploy default TF-A

The TF-A developer package provided by ST supports two toolchains:

- **Trusted boot chain** – FSBL is Trusted Firmware-A (TF-A) and SSBL is U-Boot. This is the default solution provided by ST.
- **OP-TEE** – Secure OS, i.e. a Trusted Execution Environment. This is beyond the scope of this application note but you can learn more about it [here](#).

It is recommended to build and deploy TF-A in its default form and test it before making modifications to it using the steps given under section 5.4.2 of [ST Developer Package Wiki page](#). In the compilation step, compile for several default targets using ***make -f \$PWD/./Makefile.sdk all***.

Test the trusted boot chain by using the FSBL: *tf-a-stm32mp157c-<DEVICE>_minimalconfig-mx-trusted.stm32* from build-trusted directory. The SSBL: *u-boot-stm32mp157c-dk2-trusted.stm32* is provided by U-Boot Developer Package (see Section 6). The boot log for trusted boot chain should look similar to Figure 9.

Step 3: Modifying the default TF-A source code

You can validate your toolchain and TF-A build process by adding custom log text to the TF-A source code using the example steps given under Section 8 of [How to Cross-compile ST Wiki package](#). After modifying, building and deploying TF-A on your board, the boot log should look similar to Figure 8 **Error! Reference source not found.** Please note that the actual phrase used in the example picture below is different from what is suggested in the Wiki page. Feel free to put in any phrase you desire.

```

NOTICE: CPU: STM32MP157CAC Rev.B
NOTICE: Model: STMicroelectronics STM32MP157C-DK2 STM32CubeMX board
NOTICE: Board: MB1272 Var2 Rev.C-01
INFO: Reset reason (0x15):
INFO: TF A Testing.....
INFO: Power-on Reset (rst_por)
INFO: Using SDMMC
INFO: Instance 1
INFO: Boot used partition fsbl1
INFO: Product_below_2v5=1: HSLVEN update is
INFO: destructive, no update as VDD>2.7V

```

Figure 8 Adding custom boot messages to trusted boot chain log

Step 4: Building TF-A using Custom Device Trees

- i. Copy the Custom TF-A Device Tree source files (i.e., ***stm32mp157c-<DEVICE>_minimalconfig-mx.dts*** and ***stm32mp15-mx.h***) to the ***<TF-A source code directory>/fdts/*** directory of the host Ubuntu computer.
- ii. Navigate to TF-A source directory using the command:
cd <TF-A source directory>
- iii. Build TF-A using the commands:
 - o If running **make** for the first time, use: ***make -f \$PWD/../Makefile.sdk all***
 - o After first time, use: ***make -f \$PWD/../Makefile.sdk all DEVICE_TREE="stm32mp157c-<DEVICE>_minimalconfig-mx"***
- iv. Navigate one step above ***<TF-A source directory>*** in the directory hierarchy using the command ***cd ../*** and view the contents using ***ls*** command. The build files of different boot chains will be available under ***build-trusted*** and ***build-optee*** directories.

Step 5: Configuring SD card to use the custom TF-A (Trusted boot chain)

- i. Navigate to trusted (TF-A) boot chain directory:
cd <TF-A source directory>/../build-trusted
- ii. Insert the SD card (pre-flashed with OpenSTLinux Developer image developed as part of Section 5) to the host Ubuntu computer.
- iii. Observe the partitions of the SD card by using the command: ***ls -l /dev/disk/by-partlabel/***. The part labels should look like Figure 6.
- iv. In this boot chain, only the First Stage Boot Loader (FSBL) is provided by TF-A. The SSBL is provided by U-Boot (see step 5b of Section 6). Copy only the binary (tf-a-*.stm32) to the dedicated partition named "fsbl" of the SD card using the command:
dd if=tf-a-stm32mp157c-<DEVICE>_minimalconfig-mx-trusted.stm32 of=/dev/sdb1 bs=1M conv=fdatasync
- v. Safely eject the SD card from the host computer once the transfer is complete.
- vi. Insert the SD card back into the DK2 board. Power the board. The boot log on UART4 console for the trusted tool chain should look like Figure 9.

STM32MP1 CubeMX Tutorial for OSD32MP15x

Rev.2 8/20/2019



```

NOTICE: CPU: STM32MP157CAC Rev.B
NOTICE: Model: STMicroelectronics STM32MP157C-DK2 STM32CubeMX board
NOTICE: Board: MB1272 Var2 Rev.C-01
INFO: Reset reason (0x14):
INFO: Pad Reset from NRST
INFO: Using SDMMC
INFO: Instance 1
INFO: Boot used partition fsbl1
INFO: Product_below_2v5=1: HSLVEN update is
INFO: destructive, no update as VDD>2.7V
NOTICE: BL2: v2.0(debug):
NOTICE: BL2: Built : 19:22:17, Jul 9 2019
INFO: BL2: Doing platform setup
INFO: PMIC version = 0x10
INFO: RAM: DDR3-DDR3L 16bits 533000Khz
INFO: Memory size = 0x20000000 (512 MB)
INFO: BL2 runs SP_MIN setup
INFO: BL2: Loading image id 4
INFO: Loading image id=4 at address 0x2fff0000
INFO: Image id=4 loaded: 0x2fff0000 - 0x30000000
INFO: BL2: Loading image id 5
INFO: Loading image id=5 at address 0xc0100000
INFO: STM32 Image size : 744482
WARNING: Skip signature check (header option)
INFO: Image id=5 loaded: 0xc0100000 - 0xc01b5c22
INFO: read version 0 current version 0
NOTICE: BL2: Booting BL32
INFO: Entry point address = 0x2fff0000
INFO: SPSR = 0x1d3
INFO: PMIC version = 0x10
NOTICE: SP_MIN: v2.0(debug):
NOTICE: SP_MIN: Built : 17:12:16, Jul 2 2019
INFO: ARM GICv2 driver initialized
INFO: stm32mp HSI (18): Secure only
INFO: stm32mp HSE (20): Secure only
INFO: stm32mp PLL2 (27): Secure only
INFO: stm32mp PLL2_R (30): Secure only
INFO: SP_MIN: Initializing runtime services
INFO: SP_MIN: Preparing exit to normal world

U-Boot 2018.11-stm32mp-r2 (Jul 09 2019 - 14:26:53 -0500)

CPU: STM32MP157CAC Rev.B
Model: STMicroelectronics STM32MP157C-DK2 STM32CubeMX board
Board: stm32mp1 in trusted mode (st,stm32mp157c-osd32mp157c-512m-baa_minimalconfig-mx)
Board: MB1272 Var2 Rev.C-01
DRAM: 512 MiB
Clocks:
- MPU : 650 MHz
- MCU : 208.878 MHz
- AXI : 266.500 MHz
- PER : 0 MHz
- DDR : 533 MHz
board_check_usb_power: Can't get adc device(-19)
NAND: 0 MiB
MMC: STM32 SDMMC2: 0
In: serial

```

Figure 9 - Trusted Boot Chain Boot Log

Caveat:

1. In the case of secure boot, since FSBL uses TF-A device tree and SSBL uses U-Boot device tree, make sure the **compatible property** of both these device trees are identical. Otherwise, the boot sequence may not transition from FSBL to SSBL.
2. Irrespective of the boot chain being used (basic or secure), a directory whose name corresponds to the **compatible property** of U-Boot should exist in the **bootfs** partition of SD card. U-Boot will use boot parameters defined in this directory to boot linux kernel. If U-boot fails to find this directory, the boot sequence will halt and will not transition to linux kernel. See section 9 for more information.

8 Using the Linux Device Tree

This section will guide you through the process of how to use the Linux Device Tree generated from the minimal CubeMX project. The following steps assume that you are using the **STM32MP157C-DK2** development platform from ST. It will be referred to as the “DK2 board” here after. The process of loading a custom Linux Device Tree should be similar for your custom design.

The Linux device tree generated by CubeMX consists of only one file (more info on this topic [here](#)): **stm32mp157c-<DEVICE>_minimalconfig-mx.dts**. The steps to build a custom OpenSTLinux Device Tree are as follows:

Step 1: Modifying the default Device Tree

Before replacing the default Device Tree of DK2 board (i.e., stm32mp157c-dk2.dtb) with a custom Device Tree, we will first modify the current device tree, re-compile the device tree, and verify the expected changes occurred. To do this, please complete the steps in Section 4 of the [Cross-compile with Developer Package Wiki Page](#). This process will change the default status of the on-board Green User LED on power up. By first modifying the existing Device Tree, you will be able to validate your tool chain setup and compilation procedure.

Step 2: Building a Custom Device Tree

To build your custom device tree, please complete the following steps:

- i. Copy the custom Device Tree source file (e.g., **stm32mp157c-<DEVICE>_minimalconfig-mx.dts**) to the `<kernel source code directory>/arch/arm/boot/dts/` directory of the host Ubuntu computer.
- ii. Change to the Linux kernel build directory using the command:
cd <Linux kernel build directory>
- iii. Compile the device tree (2 methods):
 - **make stm32mp157c-<DEVICE>_minimalconfig-mx** (See Caveat below)

OR

- Add the device tree name to the **Makefile** under *<kernel source code directory>/arch/arm/boot/dts/*
 - Then, navigate back to *<kernel source code directory>* and execute the command **make dtbs**
- iv. Connect the SD card to the host computer using an SD card reader. You should be able to see all the different partitions of SD card (bootfs, userfs, rootfs, vendorfs) appear under */media/<username_of_machine>*.
- v. Copy the newly built Device Tree blob (*.dtb) file to the */media/<username_of_machine>/bootfs* directory of the SD card (see Perk below):
- ```
cp <Linux kernel build directory>/arch/arm/boot/dts/stm32mp157c-
<DEVICE>_minimalconfig-mx.dtb media/<username_of_machine>/bootfs/
```

**Caveat:**

You should look at all the Device Tree nodes automatically generated by CubeMX to ensure the generated nodes reflect the configuration we have specified in the CubeMX GUI. For example, **stm32mp157c-<DEVICE>\_minimalconfig-mx.dts** has a **sound** node which is not linked to any other node. Even though no multimedia/sound interfaces are initialized in the CubeMX GUI, the **sound** node appears and must be commented out using multi-line comments as shown below:

```
/*
<sound node entries>
*/
```

Building **stm32mp157c-<DEVICE>\_minimalconfig-mx.dts** without commenting the **sound** node may result in **"Reference to non-existent node or label"** errors.

**Perk:**

While trying to write to **bootfs** directory, you may have to use **sudo** or **super user** privileges to modify permissions of **bootfs** directory since it may be write protected by default.

**Step 3: Pointing U-Boot to the Custom Device Tree:**

The final step to use the custom Device Tree is to point U-Boot to the custom device tree. To do this, go through the instructions in section 9.

## 9 Transition from U-Boot to Linux Kernel

The boot sequence of Linux on OSD32MP157C begins with the ROM code, followed by First Stage Boot Loader (FSBL), Second Stage Boot Loader (SSBL) and Linux Kernel as described [here](#). Irrespective of which boot chain is used (Basic or Secure), once U-Boot finishes its boot sequence (of FSBL and SSBL), it looks for a particular directory in */bootfs* partition of SD card corresponding to the **compatible device** defined using **compatible** property in the custom U-Boot Device Tree.

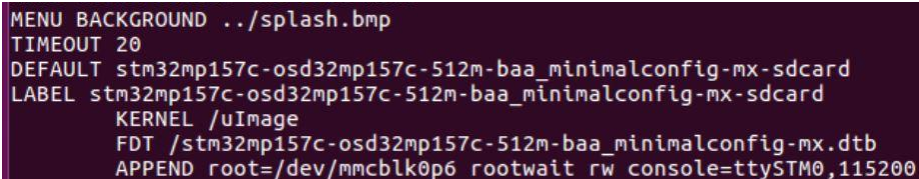
Once this directory is found, a file named **extlinux.conf** in the directory is supposed to provide information about the linux kernel device tree that needs to be loaded along with other information. If this directory is not configured properly or if the directory is not found, the boot sequence will halt. Hence it is essential to configure this directory properly. To do so, follow the below steps:

- i. Connect the SD card to the host computer using an SD card reader. You should be able to see all the different partitions of SD card (bootfs, userfs, rootfs, vendorfs) appear under */media/<username\_of\_machine>*.
- ii. Using the command line of the host PC, navigate to */media/bootfs* directory, create **mmc0\_stm32mp157c-<DEVICE>\_minimalconfig-mx\_extlinux** directory (it may be necessary to change the permissions of bootfs directory using `chmod`) and create the **extlinux.conf** file inside it using the commands:

```
cd /media/bootfs
sudo chmod 777 bootfs
mkdir mmc0_stm32mp157c-<DEVICE>_minimalconfig-mx_extlinux
vi extlinux.conf
```

- iii. Populate **extlinux.conf** file with the following content to declare the device tree that U-Boot needs to use while booting up (device tree name is declared using **FDT** tag, see Figure 10):

```
MENU BACKGROUND ../splash.bmp
TIMEOUT 20
DEFAULT stm32mp157c-<DEVICE>_minimalconfig-mx-sdcard
LABEL stm32mp157c-<DEVICE>_minimalconfig-mx-sdcard
 KERNEL /uImage
 FDT /stm32mp157c-<DEVICE>_minimalconfig-mx.dtb
 APPEND root=/dev/mmcbk0p6 rootwait rw console=ttySTM0,115200
```



```
MENU BACKGROUND ../splash.bmp
TIMEOUT 20
DEFAULT stm32mp157c-osd32mp157c-512m-baa_minimalconfig-mx-sdcard
LABEL stm32mp157c-osd32mp157c-512m-baa_minimalconfig-mx-sdcard
 KERNEL /uImage
 FDT /stm32mp157c-osd32mp157c-512m-baa_minimalconfig-mx.dtb
 APPEND root=/dev/mmcbk0p6 rootwait rw console=ttySTM0,115200
```

Figure 10 - Content of extlinux.conf file

- iv. Save the file and quit **vi editor** by pressing the **Esc** key, typing in **“:wq”**, and pressing the **Enter** key.

- v. Safely eject the SD card from the host PC and insert it back into the DK2 board.
- vi. Power the board. Once the board starts rebooting, in the UART4 serial console, you should be able to see the lines shown in Figure 11 (highlighted). These lines indicate the device tree file has been successfully loaded.

```

U-Boot SPL 2018.11-stm32mp-r2 (Jul 07 2019 - 23:58:33 -0500)
Model: STMicroelectronics STM32MP157C-DK2 STM32CubeMX board
RAM: DDR3-DDR3L 16bits 533000Khz
Trying to boot from MMC1

U-Boot 2018.11-stm32mp-r2 (Jul 07 2019 - 23:58:33 -0500)

CPU: STM32MP157CAC Rev.B
Model: STMicroelectronics STM32MP157C-DK2 STM32CubeMX board
Board: stm32mp1 in basic mode (st,stm32mp157c-osd32mp157c-512m-baa_minimalconfig-mx)
Board: MB1272 Var2 Rev.C-01
DRAM: 512 MiB
Clocks:
- MPU : 650 MHz
- MCU : 208.878 MHz
- AXI : 266.500 MHz
- PER : 0 MHz
- DDR : 533 MHz
board_check_usb_power: Can't get adc device(-19)
NAND: 0 MiB
MMC: STM32 SDMMC2: 0
In: serial
Out: serial
Err: serial
Net: No ethernet found.
Boot over mmc0!
Hit any key to stop autoboot: 0
switch to partitions #0, OK
mmc0 is current device
Scanning mmc 0:4...
Found U-Boot script /boot.scr.uimg
1553 bytes read in 1 ms (1.5 MiB/s)
Executing script at c4100000
Scanning mmc 0:4...
Found /mmc0_stm32mp157c-osd32mp157c-512m-baa_minimalconfig-mx_extlinux/extlinux.conf
Retrieving file: /mmc0_stm32mp157c-osd32mp157c-512m-baa_minimalconfig-mx_extlinux/extlinux.conf
403 bytes read in 0 ms
Retrieving file: /mmc0_stm32mp157c-osd32mp157c-512m-baa_minimalconfig-mx_extlinux/./splash.bmp
46180 bytes read in 3 ms (14.7 MiB/s)
Select the boot mode
1: stm32mp157c-osd32mp157c-512m-baa_minimalconfig-mx-sdcard
Enter choice: 1: stm32mp157c-osd32mp157c-512m-baa_minimalconfig-mx-sdcard
Retrieving file: /uImage
6569456 bytes read in 289 ms (21.7 MiB/s)
append: root=/dev/mmcblk0p6 rootwait rw console=ttySTM0,115200
Retrieving file: /stm32mp157c-osd32mp157c-512m-baa_minimalconfig-mx.dtb
65691 bytes read in 4 ms (15.7 MiB/s)
Booting kernel from Legacy Image at c2000000 ...
Image Name: Linux-4.19.9
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 6569392 Bytes = 6.3 MiB
Load Address: c2000040
Entry Point: c2000040

```

Figure 11 - Boot Log showing which .dtb file is being loaded (highlighted)

## 10 Resources

- Octavo Systems OSD32MP15x Datasheet  
<https://octavosystems.com/docs/osd32mp15x-datasheet/>
- Octavo Systems Application Note: Pin Mapping Between OSD32MP15x , the STM32MP1, and Other Integrated Devices  
[https://octavosystems.com/app\\_notes/osd32mp15x-pin-mapping-to-stm32mp1/](https://octavosystems.com/app_notes/osd32mp15x-pin-mapping-to-stm32mp1/)
- Octavo Systems Linux Device Tree Application Note  
[https://octavosystems.com/app\\_notes/osd335x-design-tutorial/osd335x-lesson-2-minimal-linux-boot/linux-device-tree/](https://octavosystems.com/app_notes/osd335x-design-tutorial/osd335x-lesson-2-minimal-linux-boot/linux-device-tree/)
- STMicroelectronics Support Community for the STM32MP157  
<https://community.st.com/s/topic/0TO0X0000003u2LWAQ/stm32mp1>

- Octavo Systems Support Forums  
<https://octavosystems.com/forums/>

If you have any additional questions using STM32MP1 CubeMX with OSD32MP15x, please contact us and work directly with our engineers! <https://octavosystems.com/contact/>.

## 11 Revision History

| Revision Number | Revision Date | Changes         | Author          |
|-----------------|---------------|-----------------|-----------------|
| 1               | 6/11/2019     | Initial Release | Eshtaartha Basu |
| 2               | 8/20/2019     | Revised Release | Eshtaartha Basu |