Linux Driver Development for Embedded Processors

NXP i.MX 7Dual Practical Labs Setup

Building a Linux Embedded System for the NXP i.MX 7Dual Processor

The i.MX 7Dual family of processors features an advanced implementation of the Arm® Cortex®-A7 core, which operates at speeds of up to 1.2 GHz, as well as the Arm Cortex-M4 core. The i.MX7Dual family supports multiple memory types including 16/32-bit DDR3L/LPDDR2/LPDDR3-1066, Quad SPI memory, NAND, eMMC, and NOR. Several high-speed connectivity connections include Gigabit Ethernet with AVB, PCIe, and USB. Both parallel and serial Display and Camera interfaces are provided, as well as a way to directly connect to the Electrophoretic Displays (EPD).

You can check all the info related to this family at https://www.nxp.com/products/processors-and-microcontrollers/applications-processors/i.mx-applications-processors/i.mx-7-processors/i.mx-7dual-processors-heterogeneous-processing-with-dual-arm-cortex-a7-cores-and-cortex-m4-core:i.MX7D

For the development of the labs the **MCIMX7SABRE**: SABRE Board for Smart Devices Based on the i.MX7Dual Applications Processors will be used. The documentation of this board can be found at https://www.nxp.com/support/developer-resources/hardware-development-tools/sabre-development-system/sabre-board-for-smart-devices-based-on-the-i.mx-7dual-applications-processors:MCIMX7SABRE

Introduction

To get the Yocto Project expected behavior in a Linux Host Machine, the packages and utilities described below must be installed. An important consideration is the hard disk space required in the host machine. For example, when building on a machine running Ubuntu, the minimum hard disk space required is about 50 GB for the X11 backend. It is recommended that at least 120 GB is provided, which is enough to compile all backends together.

Host Packages

A Yocto Project build requires that some packages be installed for the build that are documented under the Yocto Project. Please make sure your host PC is running Ubuntu 16.04 64-bit and install the following packages:

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \
build-essential chrpath socat cpio python python3 python3-pip python3-pexpect \
xz-utils debianutils iputils-ping libsdl1.2-dev xterm

$ sudo apt-get install autoconf libtool libglib2.0-dev libarchive-dev python-git \
sed cvs subversion coreutils texi2html docbook-utils python-pysqlite2 \
help2man make gcc g++ desktop-file-utils libgl1-mesa-dev libglu1-mesa-dev \
mercurial automake groff curl lzop asciidoc u-boot-tools dos2unix mtd-utils pv \
libncurses5 libncurses5-dev libncursesw5-dev libelf-dev zlib1g-dev
```

Setting up the Repo Utility

The repo tool has been developed to make it easier to manage multiple Git repositories. Instead of downloading each repository separately the repo tool can download all with one instruction. Download and install the tool by following the instructions below:

1. Create a directory for the tool. The example below creates a directory named bin in your home folder:

   ```
   $ mkdir ~/bin
   ```

2. Download the tool:

   ```
   $ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo >
   ~/bin/repo
   ```

3. Make the tool executable:

   ```
   $ chmod a+x ~/bin/repo
   ```

4. Add the directory to the PATH variable. The line below could be added to your .bashrc file so the path is available in each started shell/terminal:

   ```
   $ export PATH=~/bin:$PATH
   ```

Yocto Project Setup and Image Building

The NXP Yocto Project BSP Release directory contains a "sources" directory, which contains the recipes used to build, one or more build directories, and a set of scripts used to set up the environment.

The recipes used to build the project come from both the community and NXP. The Yocto Project layers are downloaded and placed in the sources directory. This sets up the recipes that are used to build the project.

The following example shows how to download the NXP Yocto Project Community BSP recipe layers. For this example, a directory called "imx-yocto-bsp-warrior" is created for the project:

```
~$ mkdir imx-yocto-bsp-warrior
~$ cd imx-yocto-bsp-warrior/
~/imx-yocto-bsp-warrior$ git config --global user.name "Your Name"
~/imx-yocto-bsp-warrior$ git config --global user.email "Your Email"
~/imx-yocto-bsp-warrior$ git config --list
~/imx-yocto-bsp-warrior$ repo init -u
https://source.codeaurora.org/external/imx/imx-manifest -b imx-linux-warrior -m
imx-4.19.35-1.1.0.xml
~/imx-yocto-bsp-warrior$ repo sync
```

When this process is completed, the source code is checked out into the directory imx-yocto-bsp-warrior/sources. You can perform periodic repo synchronization with the command repo sync to update to the latest code. If errors occur during repo initialization, try deleting the .repo directory and running the repo initialization command again.

There is a script, fsl-setup-release.sh, that simplifies the setup for i.MX machines. To use the script, the name of the specific machine to be built for needs to be specified as well as the desired graphical backend. The script sets up a directory and the configuration files for the specified machine and backend.

Before starting the build it must be initialized. In this step the build directory and local configuration files are created. A **distribution** must be selected when initializing the build. In our setting below the machine imx7dsabresd, the build directory build_fb and the fsl-imx-fb distribution is selected:

```
~/imx-yocto-bsp-warrior$ DISTRO=fsl-imx-fb MACHINE=imx7dsabresd source fsl-setup-
release.sh -b build-fb
```

After doing this setting you are redirected to the build-fb directory:

```
~/imx-yocto-bsp-warrior/build-fb$
```

If you are opening a new terminal before starting the build you must sourcing the setup-environment script:

```
~/imx-yocto-bsp-warrior$ source setup-environment build-fb/
```

Build now the Linux image imx-image-multimedia. This builds an i.MX image with a GUI without any Qt content.

```
~/imx-yocto-bsp-warrior/build-fb$ bitbake imx-image-multimedia
```

In the table below you can see another key images provided by Poky, meta-freescale and meta-freescale-distro:

Image name	Target	Provided by layer
core-image-minimal	A small image that only allows a device to boot.	Poky
core-image-base	A console-only image that fully supports the target device hardware.	Poky
core-image-sato	An image with Sato, a mobile environment and visual style for mobile devices. The image supports X11 with a Sato theme and uses Pimlico applications. It contains a terminal, an editor and a file manager.	Poky
fsl-image-machine-test	An FSL Community i.MX core image with console environment - no GUI interface.	meta-freescale-distro
imx-image-multimedia	Builds an i.MX image with a GUI without any Qt content.	meta-fsl-bsp-release/imx/meta-sdk
imx-image-full	Builds an opensource Qt 5 image with Machine Learning features. These images are only supported for i.MX SoC with hardware graphics. They are not supported on the i.MX 6UltraLite, i.MX 6UltraLiteLite, i.MX 6SLL, and i.MX 7Dual.	meta-fsl-bsp-release/imx/meta-sdk

When the build has finished the images will be available in the directory specified below. Please note that this directory will be different if you are using another build directory or machine configuration.

```
~$ ls imx-yocto-bsp-warrior/build-fb/tmp/deploy/images/imx7dsabresd/
```

An SD card image file .wic contains a partitioned image (with U-Boot, kernel, rootfs, etc.) suitable for booting the corresponding hardware. Extract the .wic file using the next instruction:

```
~/imx-yocto-bsp-warrior/build-fb/tmp/deploy/images/imx7dsabresd$ bunzip2 -dk -f
imx-image-multimedia-imx7dsabresd.wic.bz2
```

Finally, you are going to load the image onto an SD card. These are the instructions to program it using a laptop´s built-in SD reader:

```
~/imx-yocto-bsp-warrior/build_im7d/tmp/deploy/images/imx7dsabresd$ dmesg | tail
~/imx-yocto-bsp-warrior/build-fb/tmp/deploy/images/imx7dsabresd$ sudo dd if=imx-
image-multimedia-imx7dsabresd.wic of=/dev/mmcblk0 bs=1M conv=fsync
```

If you are using an external USB SD reader use the following commands:

```
~/imx-yocto-bsp-warrior/build-fb/tmp/deploy/images/imx7dsabresd$ sudo dd if=imx-
image-multimedia-imx7dsabresd.wic of=/dev/sdN bs=1M && sync
```

Here, /dev/sdN corresponds to the device node assigned to the SD card in your host system.

Working Outside of Yocto

You may find it more convenient to work on the kernel and develop drivers and applications outside of the Yocto Project build system. The **Yocto Project SDK** is going to help with this task. The Yocto Project SDK is:

1. A cross-compile toolchain.
2. A combination of two sysroots:
   - One for the target: Contains headers and libraries for the target. Consistent with the generated image from which it is derived.
   - One for the host: Contains host specific tools. These tools ensure things are consistent and work as expected while building against the target sysroot.
3. An environment script to setup the necessary variables to make these work together.

There are several ways to build a SDK with the Yocto Project build system:

- Using bitbake meta-toolchain. This method requires you to still install the target sysroot by installing and extracting it separately.
- Using bitbake image -c populate_sdk. This method has significant advantages over the previous method because it results in a toolchain installer that contains the sysroot that matches your target root filesystem.

Remember, before using any BitBake command, you must source the build environment setup script:

```
~/imx-yocto-bsp-warrior$ source setup-environment -b build-fb/
```

Now, build the SDK using the bitbake image -c populate_sdk option:

```
~/imx-yocto-bsp-warrior/build-fb$ bitbake -c populate_sdk imx-image-multimedia
```

When the BitBake command completes, the toolchain installer will be in tmp/deploy/sdk in the build Directory. This toolchain installer contains the sysroot that matches your target root filesystem. The resulting toolchain and matching sysroot can be installed doing:

```
~/imx-yocto-bsp-warrior/build-fb/tmp/deploy/sdk$ ./fsl-imx-fb-glibc-x86_64-imx-
image-multimedia-cortexa7hf-neon-toolchain-4.19-warrior.sh
```

To develop applications on a host machine for a different target architecture, we need a cross-compiling toolchain. For this we will use the Yocto SDK which was pre-installed in the /opt/fsl-imx-fb/4.19-warrior/ directory. Let us explore the Yocto SDK. Type the following command on the host terminal:

```
~$ tree -L 3 /opt/fsl-imx-fb/
└── 4.9-warrior
    ├── environment-setup-cortexa7hf-neon-poky-linux-gnueabi
    ├── site-config-cortexa7hf-neon-poky-linux-gnueabi
    ├── sysroots
    │   ├── cortexa7hf-neon-poky-linux-gnueabi
    │   └── x86_64-pokysdk-linux
    └── version-cortexa7hf-neon-poky-linux-gnueabi
```

Here, the 4.9-warrior folder contains scripts to export SDK environment variables. The sysroots directory contains SDK tools, libs, header files and two sub-directories, one for the host (x86_64) and one for the target (cortexa7hf). Note that you can find some information about the SDK just by reading suffixes:

- **cortexa7hf**: SDK for cortexa7 hard float (with floating point unit support).
- **neon**: neon coprocessor support.
- **linux**: for the Linux operating system.
- **gnueabi**: gnu embedded application binary interface.

To set up the SDK you have to source the environment script:

```
$ source /opt/fsl-imx-fb/4.19-warrior/environment-setup-cortexa7hf-neon-poky-
linux-gnueabi
```

This script will export several environment variables such us:

- **CC**: C compiler with target compilation options.
- **CFLAGS**: Additional C flags, used by the C compiler.
- **CXX**: C++ compiler.
- **CXXFLAGS**: Additional C++ flags, used by the CPP compiler.
- **LD**: Linker.
- **LDFLAGS**: Link flags, used by the linker.
- **GDB**: Debugger.
- **PATH:** SDK binaries added in standard command PATH.

You can look at the full set of environment variables sourced using the command below:

```
~$ export | more
```

The compiler is now in the current PATH:

```
~$ arm-poky-linux-gnueabi-gcc --version
Info (GNU textinfo) 6.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later http://gnu.org/licenses/gpl.html
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

$CC provides target gcc options:

```
~$ echo $CC
arm-poky-linux-gnueabi-gcc -mfpu=neon -mfloat-abi=hard -mcpu=cortex-a7 --
sysroot=/opt/fsl-imx-fb/4.9-warrior/sysroots/cortexa7hf-neon-poky-linux-gnueabi
```

- **float-abi**: hard: application binary interface support hard float (fpu).
- **fpu**: neon: support ARM NEON coprocessor.

- **sysroot**: where libs and header files are located.

Now, a very simple application will be compiled to verify that your toolchain is properly installed. Create the application file using for example the **gedit** text editor:

```
~$ mkdir my_first_app
~$ cd my_first_app/
~/my_first_app$ gedit app.c
```

Add the code below:

```
#include <stdio.h>
int main(void)
{
   printf("Hello World\n");
}
```

If you try to cross compile like this you will get a fatal error:

```
~/my_first_app$ arm-poky-linux-gnueabi-gcc app.c -o app
app.c:1:19: fatal error: stdio.h: No such file or directory
 #include <stdio.h>
                   ^
compilation terminated.
```

The reason is that the compiler has been configured to be generic to a wide range of ARM® processors, and the fine tuning is done when you launch the compiler with the right set of C flags. You can compile `app.c` directly with the C compiler ($CC):

```
~/my_first_app$ $CC app.c -o app
```

With the UNIX command "file" you can determine the file type (see: man file) and check the architecture and the linking method:

```
~/my_first_app$ file app
app: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked,
interpreter /lib/ld-linux-armhf.so.3,
BuildID[sha1]=6838c5cc8cfac134ee8f66a49e631263859ae6be, for GNU/LINUX 3.2.0, not
stripped
```

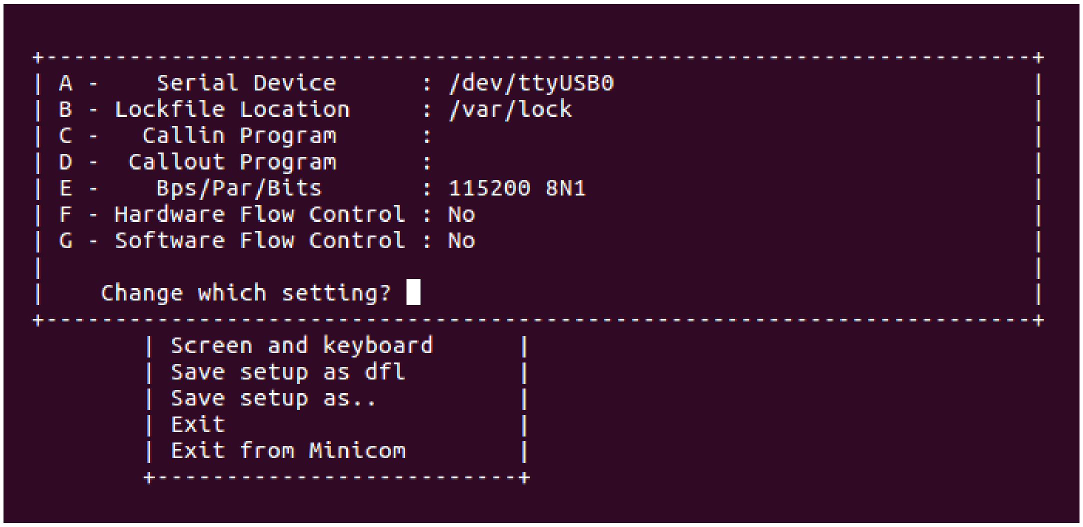
Connect and Set Up Hardware

Insert the SD with the bootable operating system image into the J6-SD Card Socket of the MCIMX7SABRE board. The following table shows the boot switch settings (which takes priority in boot process) for booting from the SD card on the i.MX7Dual Sabre board.

Boot Media	SW2 [D1-D8]	SW3 [D1-D2]
SD card (SD1)	00100000	10
eMMC	01010000	10
NAND	011XXXX0	10
QuadSPI	10000000	10
SDP	XXXXXXXX	01

Connect the MCIMX7SABRE board to your host computer via the J11-Debug UART connector using a micro USB cable. Open a serial console using a serial port communication program, for example **minicom**. Through this console, you can access the MCIMX7SABRE running an embedded Linux distribution. Launch and configure minicom in your host to see the booting of the system. Set the following configuration: "115.2 kbaud, 8 data bits, 1 stop bit, no parity".

```
~$ sudo minicom -w -s
```



```
+-----------------------------------------------------------------------+
| A -    Serial Device      : /dev/ttyUSB0                              |
| B - Lockfile Location     : /var/lock                                 |
| C -   Callin Program      :                                           |
| D -  Callout Program      :                                           |
| E -    Bps/Par/Bits       : 115200 8N1                                |
| F - Hardware Flow Control : No                                        |
| G - Software Flow Control : No                                        |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
        | Screen and keyboard      |
        | Save setup as dfl        |
        | Save setup as..          |
        | Exit                     |
        | Exit from Minicom        |
        +--------------------------+
```

Connect the 5V power supply cable to the 5V DC power jack (J1). Power the board by flipping the switch (SW1). The processor starts executing from the on-chip ROM code. With the default boot switch setup, the code reads the fuses to define the media where it is expected to have a bootable image. After it finds a bootable image on the SD card, the U-Boot execution should begin automatically.

Information is printed in the smaller number serial console for the Cortex-A7 (COM9 on Windows as an example and /dev/ttyUSB* on Linux). If you don not stop the U-Boot process, it continues to boot the Linux kernel. Once Linux is booted, login using the user name root and no password.

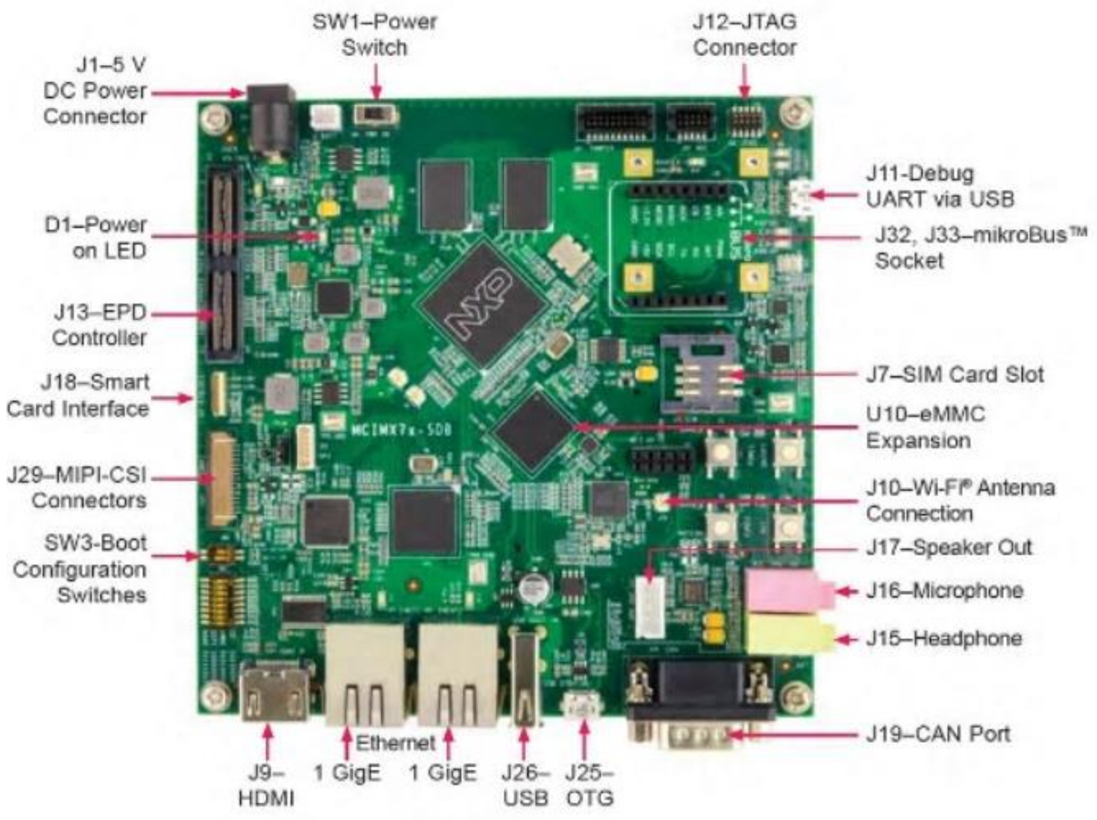


Figure 1. i.MX7Dual Sabre front

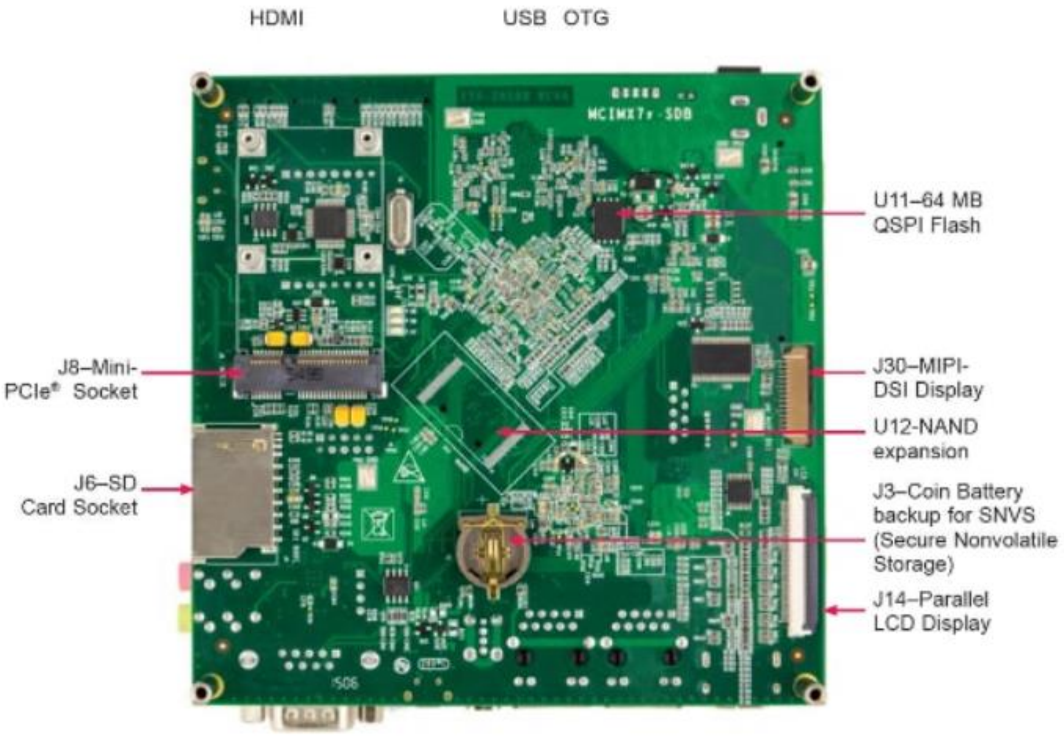


Figure 2. i.MX7Dual Sabre back

Establish a network connection between the Linux host and the MCIMX7SABRE board so that drivers built on the Linux host PC can be easily transferred to the target for installation. Connect the Ethernet cable between your host PC and the MCIMX7SABRE board Ethernet port shown on the left side (close to the J9-HDMI) of the attached "i.MX7Dual Sabre front" image. Set up the Linux host PC's IP Address to **10.0.0.1**. On the MCIMX7SABRE board (the target), configure the eth0 interface with IP address **10.0.0.10** by editing the /etc/network/interfaces file. Open the file using vi editor:

```
root@imx7dsabresd:~# vi /etc/network/interfaces
```

Start editing the file by hitting **"i"**. You may find there is already an entry for eth0 as shown below. If not, configure eth0 with the following lines:

```
auto eth0
iface eth0 inet static
address 10.0.0.10
```

```
netmask 255.255.255.0
```

Exit the editing mode by hitting the ESC button. Save and exit the file by typing **":wq"**. If any time you want to exit the file without saving the changes you made, hit ESC followed by: **":q!".**

Then, run the following commands to make sure that the eth0 Ethernet interface is properly configured:

```
root@imx7dsabresd:~# ifdown eth0
root@imx7dsabresd:~# ifup eth0
root@imx7dsabresd:~# ifconfig
```

Now, test that you can communicate with your Linux host machine from your MCIMX7SABRE board. Exit the ping command by hitting **ctrl-C**.

```
root@imx7dsabresd:~# ping 10.0.0.1
```

A network connection is now established between the Linux host and the MCIMX7SABRE target.

Building the Linux Kernel

The kernel will be configured and built outside of the Yocto build system. Download the kernel source code from the Code Aurora git repositories:

```
~$ git clone https://source.codeaurora.org/external/imx/linux-imx -b
imx_4.19.35_1.1.0
```

Source the SDK environment script:

```
$ source /opt/fsl-imx-fb/4.19-warrior/environment-setup-cortexa7hf-neon-poky-
linux-gnueabi
```

Prior to compiling the Linux kernel it is often a good idea to make sure that the kernel sources are clean and that there are no remnants left over from a previous build:

- **clean** - Remove most generated files but keep the config and enough build support to build external modules.
- **mrproper** - Remove all generated files + config + various backup files.
- **distclean** - mrproper + remove editor backup and patch files.

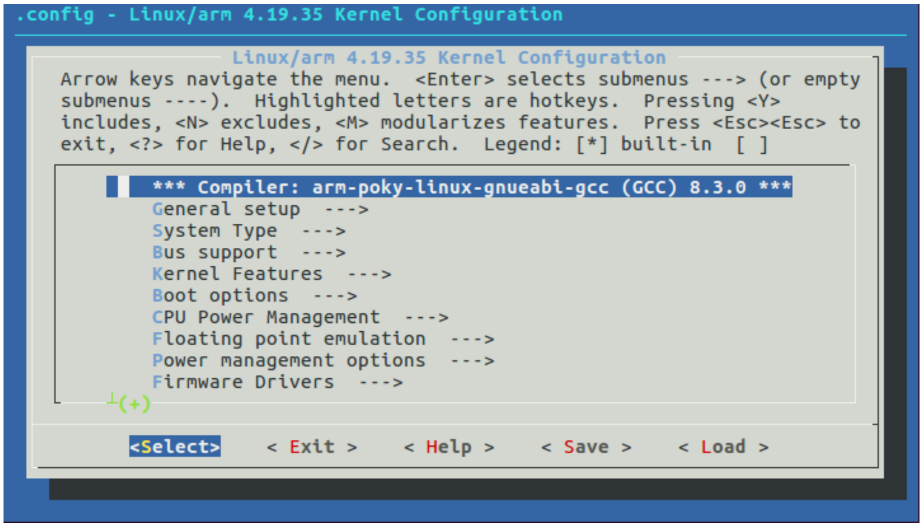
```
~/linux-imx$ make mrproper
```

It is often easiest to start with a base default configuration and then customize it for your use case if needed. The **imx_v7_defconfig** located in arch/arm/configs will be used as a starting point:

```
~/linux-imx$ make imx_v7_defconfig
```

When you want to customize the kernel configuration the easiest way is to use the built-in kernel configuration systems. One of the most popular configuration systems is the **menuconfig** utility.

```
~/linux-imx$ make menuconfig
```



```
.config - Linux/arm 4.19.35 Kernel Configuration
                    Linux/arm 4.19.35 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
  submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
  exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]

        *** Compiler: arm-poky-linux-gnueabi-gcc (GCC) 8.3.0 ***
        General setup  --->
        System Type  --->
        Bus support  --->
        Kernel Features  --->
        Boot options  --->
        CPU Power Management  --->
        Floating point emulation  --->
        Power management options  --->
        Firmware Drivers  --->
      (+)

     <Select>    < Exit >    < Help >    < Save >    < Load >
```

Configure the following kernel settings that will be needed during the development of your drivers:

```
Device drivers >
   [*] SPI support  --->
          <*>   User mode SPI device driver support

Device drivers >
   [*] LED Support  --->
           <*>   LED Class Support
           -*-   LED Trigger support  --->
                        <*>   LED Timer Trigger
                        <*>   LED Heartbeat Trigger

Device drivers >
   <*> Industrial I/O support  --->
          -*-   Enable buffer support within IIO
          -*-   Industrial I/O buffering based on kfifo
          <*>  Enable IIO configuration via configfs
          -*-   Enable triggered sampling support
```

```
            <*>   Enable software IIO device support
            <*>   Enable software triggers support
                   Triggers - standalone  --->
                           <*> High resolution timer trigger
                           <*> SYSFS trigger

    Device drivers >
        <*> Userspace I/O drivers  --->
            <*>   Userspace I/O platform driver with generic IRQ handling

    Device drivers >
       Input device support  --->
            -*- Generic input layer (needed for keyboard, mouse, ...)
            <*>   Polled input device skeleton
```

Save the configuration and exit from menuconfig.

Once the kernel has been configured it must be compiled to generate the bootable kernel image as well as any dynamic kernel modules that were selected. By default U-Boot expects **zImage** to be the type of kernel image used.

```
    ~/linux-imx$ make zImage
```

Starting with Linux kernel version 3.8 each ARM® board has an unique device tree binary file required by the kernel. Therefore, you will need to build and install the correct .dtb for the target device. All device tree files are located under arch/arm/boot/dts/. To build an individual device tree file find the name of the .dts file for the board you are using and replace the .dts extension with .dtb. The compiled device tree file will be located under arch/arm/boot/dts/. Then, run the following command to compile an individual device tree file:

```
~/linux-imx$ make imx7d-sdb.dtb
```

To build all the device tree files:

```
~/linux-imx$ make dtbs
```

By default the majority of the Linux drivers are not integrated into the kernel image (e.g., zImage). These drivers are built as dynamic modules. This will result in .ko (kernel object) files being placed in the kernel tree. These .ko files are the dynamic kernel modules. Whenever you make a change to the kernel its generally recommended that you rebuild your kernel modules and then reinstall the kernel modules. Otherwise the kernel modules may not load or run.

```
~/linux-imx$ make modules
```

Install the modules; insert the SD into a laptop´s built-in SD reader:

```
    ~$ lsblk
    ~$ mkdir ~/mnt
```

```
~$ mkdir ~/mnt/ext4
~$ sudo mount /dev/mmcblk0p2 ~/mnt/ext4/
~$ ls -l ~/mnt/ext4/ /* see the files in the ext4 partition */
```

Install the modules:

```
~/linux-imx$ make modules_install INSTALL_MOD_PATH=~/mnt/ext4/
~$ sudo umount ~/mnt/ext4
```

To compile the kernel image, modules, and all the device tree files in a single step:

```
~/linux-imx$ make -j8
```

Once the Linux kernel, dtb files and modules have been compiled they must be installed. In the case of the kernel image this can be installed by copying the zImage file to the location where it will be read from. There are two partitions on the SD card. The smaller one, formatted as a FAT file system, holds zImage and imx7d-sdb.dtb. The larger partition, formatted as ETX4, contains a root file system. The SD card as whole is represented among the devices as /dev/mmcblk0, its partitions as /dev/mmcblk0p1 and /dev/mmcblk0p2 respectively. The second partition is mounted as a root on the target. The boot partition is not mounted. We must mount it first before we can access it.

```
root@imx7dsabresd:~# mkdir /mnt/linux
root@imx7dsabresd:~# mount -t vfat /dev/mmcblk0p1 /mnt/linux/
root@imx7dsabresd:~# ls -l /mnt/linux/
total 8806
-rwxrwx--- 1 root disk   48323 Mar  4  2020 imx7d-sdb-epdc.dtb
-rwxrwx--- 1 root disk   48002 Mar  4  2020 imx7d-sdb-gpmi-weim.dtb
-rwxrwx--- 1 root disk   48192 Mar  4  2020 imx7d-sdb-m4.dtb
-rwxrwx--- 1 root disk   48332 Mar  4  2020 imx7d-sdb-mipi-dsi.dtb
-rwxrwx--- 1 root disk   48425 Mar  4  2020 imx7d-sdb-qspi.dtb
-rwxrwx--- 1 root disk   48591 Mar  4  2020 imx7d-sdb-reva-epdc.dtb
-rwxrwx--- 1 root disk   48270 Mar  4  2020 imx7d-sdb-reva-gpmi-weim.dtb
-rwxrwx--- 1 root disk   48038 Mar  4  2020 imx7d-sdb-reva-hdmi-audio.dtb
-rwxrwx--- 1 root disk   48460 Mar  4  2020 imx7d-sdb-reva-m4.dtb
-rwxrwx--- 1 root disk   48693 Mar  4  2020 imx7d-sdb-reva-qspi.dtb
-rwxrwx--- 1 root disk   48262 Mar  4  2020 imx7d-sdb-reva-touch.dtb
-rwxrwx--- 1 root disk   48038 Mar  4  2020 imx7d-sdb-reva-wm8960.dtb
-rwxrwx--- 1 root disk   48262 Mar  4  2020 imx7d-sdb-reva.dtb
-rwxrwx--- 1 root disk   47994 Mar  4  2020 imx7d-sdb.dtb
-rwxrwx--- 1 root disk   15620 Mar  4  2020 imx7d_sabresd_m4_TCM_Pingpang.bin
-rwxrwx--- 1 root disk   10464 Mar  4  2020 imx7d_sabresd_m4_TCM_helloworld.bin
-rwxrwx--- 1 root disk   16236 Mar  4  2020 imx7d_sabresd_m4_TCM_mcctty.bin
-rwxrwx--- 1 root disk  271808 Mar  4  2020 tee.bin
-rwxrwx--- 1 root disk  271872 Mar  4  2020 uTee-7dsdb
-rwxrwx--- 1 root disk 7738352 Mar  4  2020 zImage

root@imx7dsabresd:~#
```

Copy the imx7d-sdb.dtb and the zImage files from the host to target's /mnt/linux directory. We use **scp** (secure copy) program. We copy the file as root, because we logged in to the target as root. Enter the commands on the host terminal. When a secure copy program runs for the first time, it will notice it is communicating with an unknown machine. It will report the fact. Accept it. The host PC will add the target to the list of known machines.

```
~/linux-imx$ scp arch/arm/boot/zImage root@10.0.0.10:/mnt/linux
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
SHA256:OT7AUSpGwSSNkzbH+WO1xUOW/UaKFURIHg3iG2YdvsA.
Please contact your system administrator.
Add correct host key in /home/alberto/.ssh/known_hosts to get rid of this message.
Offending RSA key in /home/alberto/.ssh/known_hosts:2
  remove with:
  ssh-keygen -f "/home/alberto/.ssh/known_hosts" -R 10.0.0.10
RSA host key for 10.0.0.10 has changed and you have requested strict checking.
Host key verification failed.
lost connection

~/linux-imx$ ssh-keygen -f "/home/alberto/.ssh/known_hosts" -R 10.0.0.10
~/linux-imx$ scp arch/arm/boot/zImage root@10.0.0.10:/mnt/linux
The authenticity of host '10.0.0.10 (10.0.0.10)' can't be established.
RSA key fingerprint is SHA256:OT7AUSpGwSSNkzbH+WO1xUOW/UaKFURIHg3iG2YdvsA.
Are you sure you want to continue connecting (yes/no)? Yes
Warning: Permanently added '10.0.0.10' (RSA) to the list of known hosts.
zImage                                        100% 7565KB   7.4MB/s   00:01

~/linux-imx$ scp arch/arm/boot/dts/imx7d-sdb.dtb root@10.0.0.10:/mnt/linux
root@imx7dsabresd:~# umount /mnt/linux
```

Reboot the MCIMX7SABRE board:

```
root@imx7dsabresd:~# reboot
```

Download the linux_4.19_imx7_drivers file from the github of the book and unzip it in the home folder of the Linux host:

```
~$ cd ~/linux_4.19_imx7_drivers/
```

Compile and the deploy the drivers to the MCIMX7SABRE board:

```
~/linux_4.19_imx7_drivers$ make
~/linux_4.19_imx7_drivers$ make deploy
scp *.ko root@10.0.0.10:
```

```
adxl345_imx.ko                                        100%   12KB  12.3KB/s   00:00
adxl345_imx_iio.ko                                    100%   12KB  12.4KB/s   00:00
hellokeys_imx.ko                                      100% 7024     6.9KB/s   00:00
helloworld_imx.ko                                     100% 4008     3.9KB/s   00:00
helloworld_imx_char_driver.ko                         100% 6184     6.0KB/s   00:00
helloworld_imx_class_driver.ko                        100% 7724     7.5KB/s   00:00
helloworld_imx_with_parameters.ko                     100% 4604     4.5KB/s   00:00
helloworld_imx_with_timing.ko                         100% 5688     5.6KB/s   00:00
i2c_imx_accel.ko                                      100% 7216     7.1KB/s   00:00
int_imx_key.ko                                        100% 7812     7.6KB/s   00:00
int_imx_key_wait.ko                                   100%   10KB   9.9KB/s   00:00
io_imx_expander.ko                                    100% 9664     9.4KB/s   00:00
keyled_imx_class.ko                                   100%   16KB  16.2KB/s   00:00
ledRGB_imx_class_platform.ko                          100% 9524     9.3KB/s   00:00
ledRGB_imx_platform.ko                                100%   11KB  10.9KB/s   00:00
led_imx_UIO_platform.ko                               100% 6912     6.8KB/s   00:00
linkedlist_imx_platform.ko                            100% 9460     9.2KB/s   00:00
ltc2422_imx_dual.ko                                   100% 7344     7.2KB/s   00:00
ltc2422_imx_trigger.ko                                100% 9840     9.6KB/s   00:00
ltc2607_imx_dual_device.ko                            100% 8056     7.9KB/s   00:00
ltc3206_imx_led_class.ko                              100%   11KB  11.1KB/s   00:00
misc_imx_driver.ko                                    100% 5780     5.6KB/s   00:00
sdma_imx_m2m.ko                                       100%   12KB  11.7KB/s   00:00
sdma_imx_mmap.ko                                      100%   12KB  11.7KB/s   00:00

~/linux_4.19_imx7_drivers$
```

Verify that the drivers are now in the MCIMX7SABRE board:

```
root@imx7dsabresd:~# ls
adxl345_imx.ko                     keyled_imx_class.ko
adxl345_imx_iio.ko                 ledRGB_imx_class_platform.ko
hellokeys_imx.ko                   ledRGB_imx_platform.ko
helloworld_imx.ko                  led_imx_UIO_platform.ko
helloworld_imx_char_driver.ko      linkedlist_imx_platform.ko
helloworld_imx_class_driver.ko     ltc2422_imx_dual.ko
helloworld_imx_with_parameters.ko  ltc2422_imx_trigger.ko
helloworld_imx_with_timing.ko      ltc2607_imx_dual_device.ko
i2c_imx_accel.ko                   ltc3206_imx_led_class.ko
int_imx_key.ko                     misc_imx_driver.ko
int_imx_key_wait.ko                sdma_imx_m2m.ko
io_imx_expander.ko                 sdma_imx_mmap.ko

root@imx7dsabresd:~#
```

The imx7d-sdb.dts file with all the needed modifications to run the drivers is stored in the device_tree folder inside the linux_4.19_imx7_drivers folder. You can replace the imx7-sdb.dts file included in the ~/linux-imx/arch/arm/boot/dts/ folder for this new .dts one, then compile it and deploy the new imx7-sdb.dtb file to the MCIMX7SABRE board.

During the development of the LABs 5.2, 5.3, 5.4 you have to disable the ecspi3 node, as the two pads in bold below are required for the drivers developed during these LABs:

```
&ecspi3 {
   fsl,spi-num-chipselects = <1>;
   pinctrl-names = "default";
   pinctrl-0 = <&pinctrl_ecspi3 &pinctrl_ecspi3_cs>;
   cs-gpios = <&gpio5 9 GPIO_ACTIVE_HIGH>, <&gpio6 22 0>;
   status = "disabled";

pinctrl_ecspi3: ecspi3grp {
                  fsl,pins = <
                        MX7D_PAD_SAI2_TX_SYNC__ECSPI3_MISO   0x2
                        MX7D_PAD_SAI2_TX_BCLK__ECSPI3_MOSI   0x2
                        MX7D_PAD_SAI2_RX_DATA__ECSPI3_SCLK   0x2
                  >;
};
```

During the development of the LABs 7.1, 7.2, 7.3 you have to comment out the gpio-keys node, as there are some conflicts with the GPIOs used for the drivers developed during these LABs:

```
/*gpio-keys {
          compatible = "gpio-keys";
          pinctrl-names = "default";
          pinctrl-0 = <&pinctrl_gpio_keys>;

          volume-up {
                label = "Volume Up";
                gpios = <&gpio5 11 GPIO_ACTIVE_LOW>;
                linux,code = <KEY_VOLUMEUP>;
          };

          volume-down {
                label = "Volume Down";
                gpios = <&gpio5 10 GPIO_ACTIVE_LOW>;
                linux,code = <KEY_VOLUMEDOWN>;
          };
   };*/
```

During the development of the LABs 10.2, 11.2, 11.3, 11.4, 12.1 you have to enable the ecspi3 node, as this SPI controller is used for the drivers developed during these LABs:

```
&ecspi3 {
   fsl,spi-num-chipselects = <1>;
   pinctrl-names = "default";
   pinctrl-0 = <&pinctrl_ecspi3 &pinctrl_ecspi3_cs>;
```

```
        cs-gpios = <&gpio5 9 GPIO_ACTIVE_HIGH>, <&gpio6 22 0>;
        status = "okay";
```

The LAB 9.2 has not been ported to this kernel 4.19 version, as DMA_SG is not supported in the Linux kernel anymore.