Rev: 2.0.1_2
Date: 19/11/2019

TIXU_MX8M User's Guide

Table of contents

1. Introduction

This document provides the complete details to work with iMX8M Mini EVK powered by TI PMICs TPS65218D0 and LP8733(hereafter specifying as TIXU_MX8M). Figure 1 and Figure 2 shows the top view and bottom view of EVK.

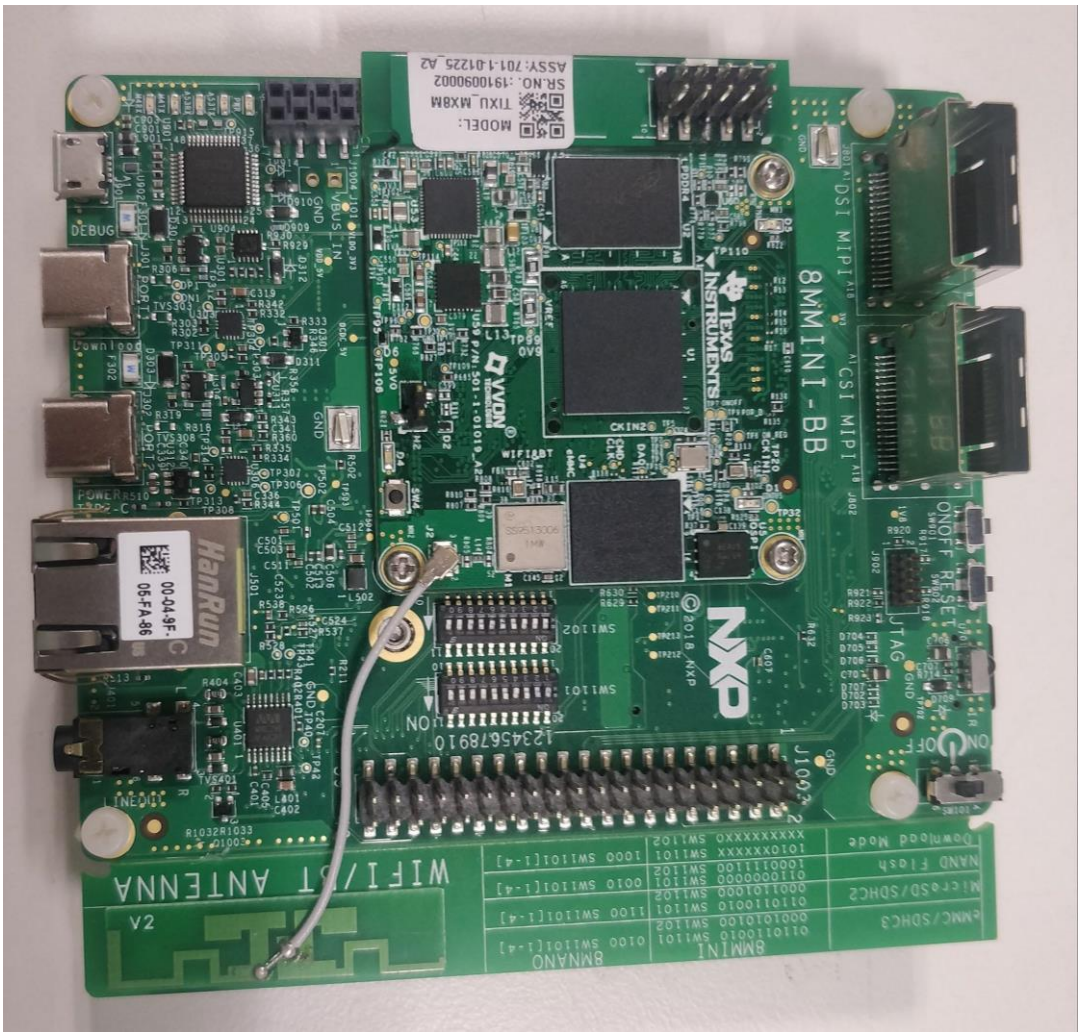

Figure 1 : Top view of EVK

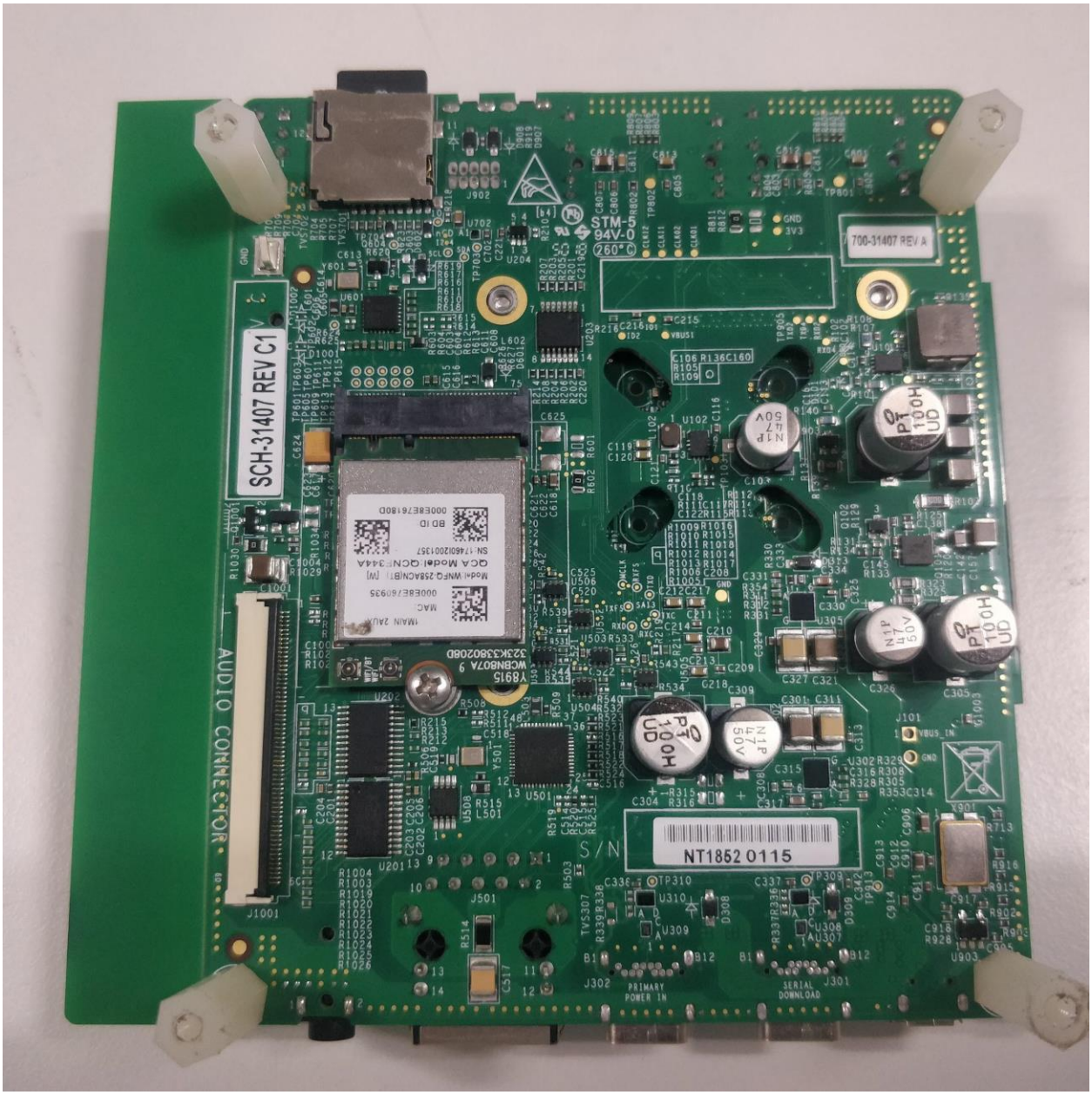

Figure 2 : Bottom view of EVK

2. Preparing SD card for TIXU_MX8M EVK

The primary boot source we provided for TIXU_MX8M EVK is SD card. Use either prebuilt executables or build new executables to prepare SD card. Following section describes the details to prepare SD card either form newly compiled images(after compiling source code from user side) or from prebuilt images.

2.1. Setup SDK and create executables

This section will guide the user to compile the source code and create executables from their end. Following are the prerequisites:-

System requirements are:-

- Ubuntu 16.04
- 120GB HDD
- TIXU_MX8M EVK
- micro SD Card
- micro USB cable
- 5 V supply

Yocto build requirements are:-

- Essentials: Packages needed to build an image on a headless system:
  $ 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

- Graphical and Eclipse Plug-In Extras: Packages recommended if the host system has graphics support or if you are going to use the Eclipse IDE:
  $ sudo apt-get install libsdl1.2-dev xterm

- Documentation: Packages needed if you are going to build out the Yocto Project documentation manuals:
  $ sudo apt-get install make xsltproc docbook-utils fop dblatex xmlto

- OpenEmbedded Self-Test (oe-selftest): Packages needed if you are going to run oe-selftest:
  $ sudo apt-get install python-git

Install above packages in the host pc to compile the source code successfully.

To setup and compile SDK, download the source code from the provided VVDN gitlab link. Recommended host system for compilation is Ubuntu 16.04. Click on the link provided here to download i.MX Yocto Project User's Guide and refer 3.1 Host packages to install yocto specific packages. Follow below steps from host PC to download and compile the source code:-

- Make sure that git is set up properly with the commands below
  $ git config --global user.name "Your Name"
  $ git config --global user.email "Your Email"
  $ git config --list

- download the i.MX Yocto Project Community BSP recipe layers
  $ git clone https://gitlab.vvdntech.com:8081/tixu_mx7d/tixu_mx7d.git
  $ git checkout 8mmini_release

- Enter in to base directory ie, imx-yocto-bsp
  $ cd <path_to_source_code>/imx-yocto-bsp

- Use below command to setup build configurations.

$ DISTRO=fsl-imx-xwayland MACHINE=timx8m source fsl-setup-release.sh -b build

MACHINE= <machine configuration name> is the machine name which points to the configuration file in conf/machine in meta-freescale and meta-fsl-bsp-release. New machine name added for TIXU_MX8M to identify changes made by VVDN. So use MACHINE=timx8m in the build configuration to compile sdk for TIXU_MX8M.
-b <build dir> specifies the name of the build directory created by the fsl-setup-release.sh script.
Refer 5.1 of i.MX Yocto Project User's Guide for more details.

- After executing the above command, it will redirect to the build directory specified in the command. Then use below command to compile-
  $ bitbake fsl-image-validation-imx

  First compilation will take more time to finish.

- After successful compilation, executables will present in below location in build directory:-
  tmp/deploy/images/timx8m/

  Below are the executables(After compilation),

fsl-image-validation-imx-timx8m-<date_time>.rootfs.sdcard.bz2 - single image containing u-boot, kernel and file system.
u-boot.bin - u-boot
Image - kernel image
Image-timx8m.dtb - kernel device tree
fsl-image-validation-imx-timx8m-<date and time>.rootfs.tar.bz2 - file system.

- Follow below steps to flash single image(*.sdcard.bz2) into SD card-
  $bunzip2 -dk -f fsl-image-validation-imx-timx8m-<date_time>.rootfs.sdcard.bz2 (replace date_and_time according to compilation time)
  $ sudo dd if=fsl-image-validation-imx-timx8m-<date_time>.rootfs.sdcard of=/dev/sd<device_node> bs=1M conv=fsync

  Here device_node is the name of the node that represents the device to which image is copied. It can be sdb, sdc, mmcblk0, mmcblk1 etc.(replace date_time according to compilation time).
  $ sync

  Now the SD card is ready for booting TIXU_MX8M EVK.

- Compile u-boot, kernel and complete SD card image
  1. Compile u-boot using below commands:-
     build$ bitbake -f -c compile u-boot-imx
     build$ bitbake u-boot-imx

  2. Compile kernel using below commands:-
     build$ bitbake -f -c compile linux-imx
     build$ bitbake linux-imx

  3. Do full compilation to create single image(in *.sdcard.bz2)
     build$ bitbake fsl-image-validation-imx

- Now the image is ready to flash to SD card, to boot TIXU_MX8M EVK. Then follow below commands to flash images into SD card(verify date and time of image before flashing to sd card).

  $ cd tmp/deploy/images/timx8m/

  $ bunzip2 -dk -f fsl-image-validation-imx-timx8m-<date_time>.rootfs.sdcard.bz2 (replace date_time according to compilation time)

$ sudo dd if=fsl-image-validation-imx-timx8m-<date_time>.rootfs.sdcard of=/dev/sd<device_node> bs=1M conv=fsync.

(replace date_time according to compilation time and device_node with name of node that representing the device to which image is copying.)

$ sync

Now the SD card is ready to boot EVK.

2.2. Flashing image from precompiled binaries

The precompiled image is present in the tar file which is shared by VVDN. Following are the requirements to flash precompiled image in to SD card.

System requirements are:-

- PC (ubuntu 16.04)
- TIXU_MX8M EVK
- micro SD card 4GB
- SD card Adapter / Card Reader
- micro USB cable
- 5V power supply

Required tools are:-

- bunzip2
- dd

Follow below steps to prepare bootable SD card:-

- Download release tar file(shared by VVDN, with name v2.0.1_2) to get prebuilt images.
- Extract the compressed tar file. After extracting, we will get one folder containing all documents and prebuilt image.
- Use prebuilt image to prepare bootable SD card.
  eg:- *fsl-image-validation-imx-timx8m-20190904153451.rootfs.sdcard.bz2*
- Extract the bz2 image using bunzip2. It will give a sd card image
  $ bunzip2 -dk -f fsl-image-validation-imx-timx8m-<date_time>.rootfs.sdcard.bz2
- Insert microSD card to SD card Adapter / Card Reader

- Insert the SD card adapter to SD card port of laptop or Card reader to USB port

- flash the image to SD card using below commands
  $sudo dd if= <fsl-image-validation-imx-timx8m-<date_time>.rootfs.sdcard> of=/dev/<dev_name> bs=1M conv=fsync

  Change dev_name based on the device node. It can sdb, sdc, mmcblk0 etc. Wait some time to finish sd card programming.

  $ sync

- Now SD card is ready for booting TIXU_MX8M EVK.

2.3 Flashing the eMMC or SD card partition using uuu tool

1. After cloning the image from gitlab, checking out to that tag v2.0.1_2 and finished compiling the full image execute the following command to copy the uuu tool to the images folder.
   $ cp ../sources/flashing_tool/uuu/uuu tmp/deploy/images/timx8m/
2. Put the board in the serial download mode and connect the 'Download type-C' port of the board to the linux system.
3. Execute the following command to flash the image to the eMMC:-
   sudo ./uuu -b emmc_all imx-boot-timx8m-sd.bin-flash_evk\
   fsl-image-validation-imx-timx8m-<date_time>.rootfs.sdcard
4. Execute the following command to flash image to the sd_card:-
   sudo ./uuu -b sd_all imx-boot-timx8m-sd.bin-flash_evk\
   fsl-image-validation-imx-timx8m-<date_time>.rootfs.sdcard
5. After successful flashing we can change the boot mode switches to boot from SD card or eMMC and verify the flashed image.

3. Booting of TIXU_MX8M EVK

The primary booting option we provided is SD card. To flash images in to SD card, follow section 2.1 or section 2.2. Insert the SD card into the SD card slot provided in the EVK and set the boot switches(Refer installation guide) to boot from SD card(if executables are not found in the configured boot source, then it automatically fetches executables from SD card). Connect micro USB to debug port of EVK and other end to host PC. make connections as shown in the below figure.

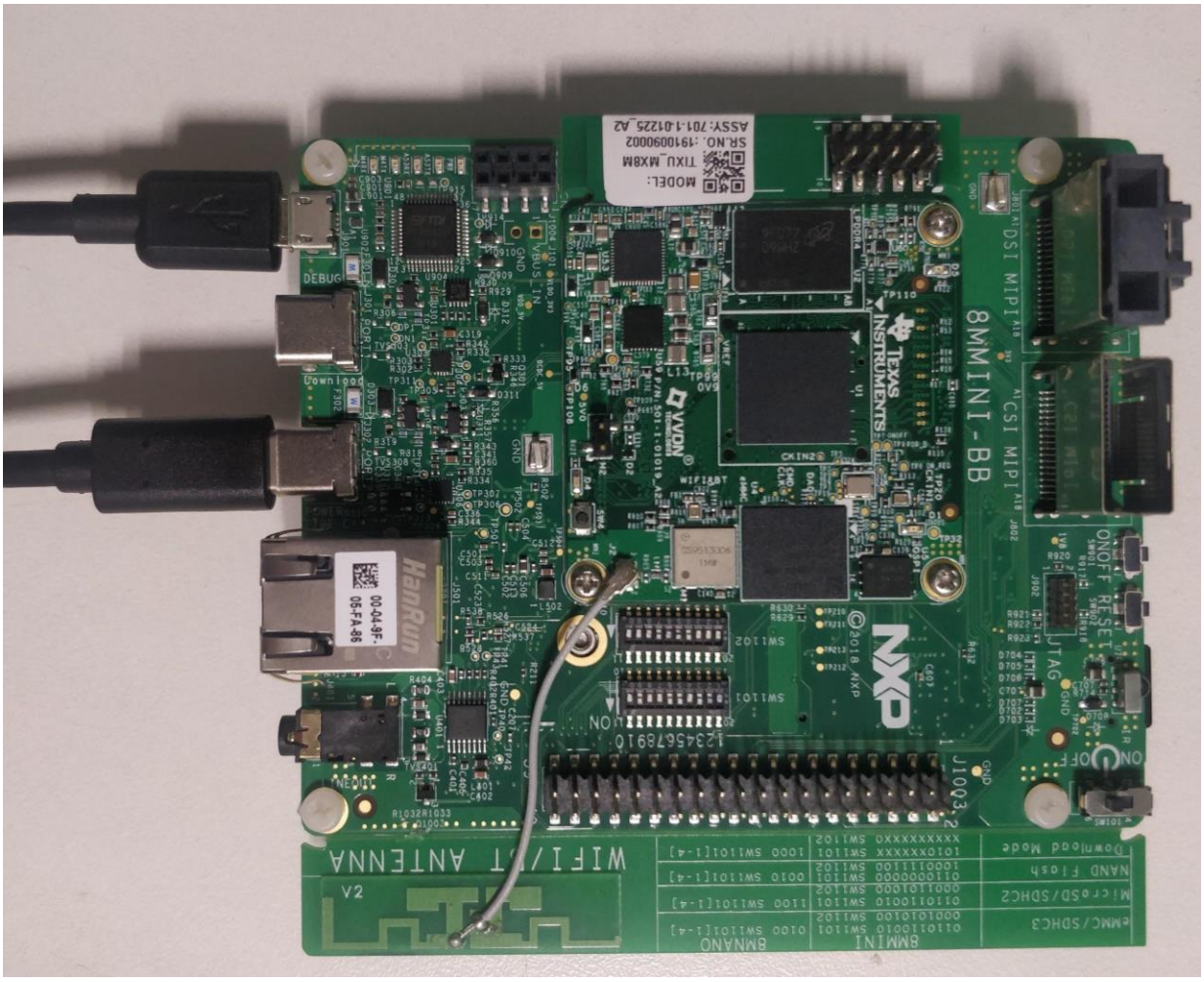

Figure 3 : Board setup to verify booting

Use teraterm or putty to get debug log from the device node if the host pc is windows. Figure 4 and Figure 5 shows the settings for teraterm to get debug logs (Change port number accordingly):

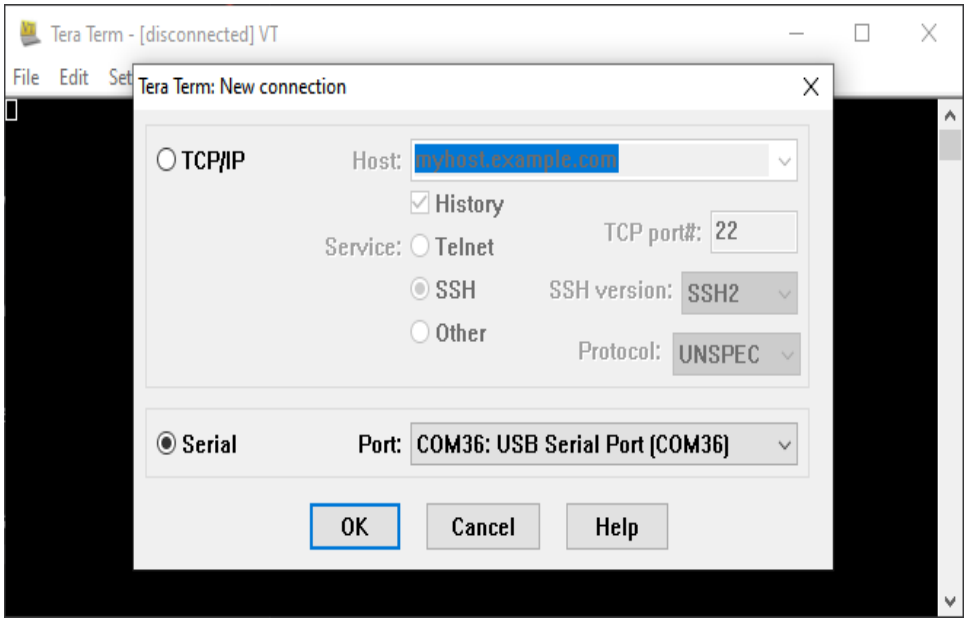

Figure 4 : Opening new teraterm

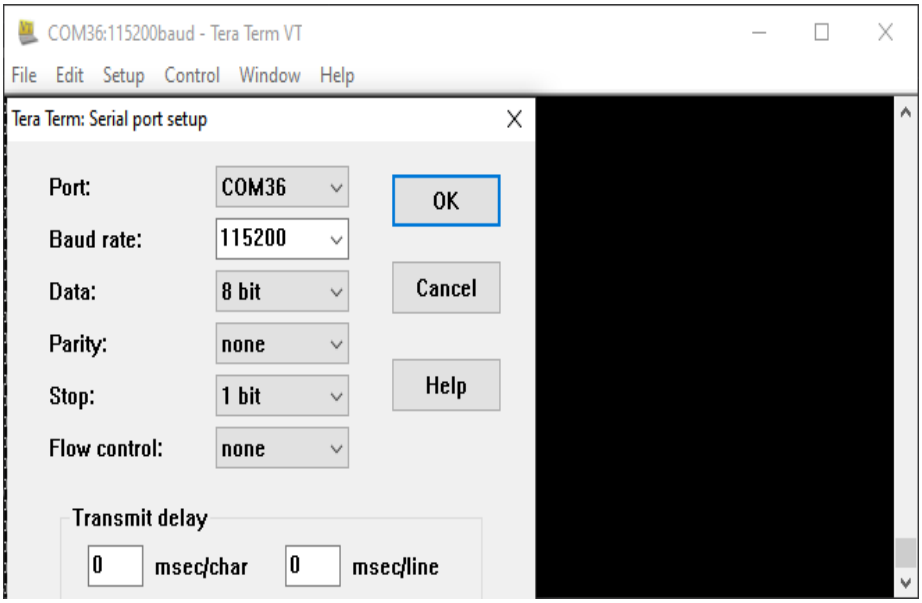

Figure 5 : Teraterm settings

If the host PC is Ubuntu, then use below commands to verify booting of EVK

$ sudo minicom -s

Refer section 4.1 to change serial port configurations. Change required serial configurations, then save and exit. Now it waits for data from serial port. Give power to EVK by changing the state of switches SW1101 and SW1102 after making the setup ready.

4. Interface verification

This section provides steps to verify the interfaces in EVK(verified with ubuntu PC).

4.1. Debug uart testing

Follow below steps to verify working of debug uart

1. Connect USB 2.0 Cable from Debug Board to Laptop.
2. Connect the Power Jack to the target Board.
3. Set below minicom Configurations after running 'sudo minicom -s' in terminal of host PC(ubuntu):-
   * Select Serial Port setup(A) - Change the serial device to respective Device node.
     -> Device Node can be checked by running dmesg in host laptop
     -> open another terminal run dmesg | grep tty
     -> <FTDI USB Serial Device converter now attached to ttyUSB1> will get displayed
     -> Now <Device_Node> is /dev/ttyUSB1 (Go back to minicom terminal)
     -> Enter Serial Device as <Device_node>
   * Select <Bps/Par/Bits> by entering Required Option(E)
     -> Configure speed to 115200
     -> Configure data as 8 parity as NONE and stop bit as 1
   * Select Hardware Flow Control by entering required option(F)
     -> Set Hardware Flow Control as No
   * Select Software Flow Control by entering required option(G)
     -> Set Software Flow Control as No
   * Then press enter and select "Save setup as dfl"
     * select exit, now minicom is ready.
4. Then boot the board and verify logs on the console.

If the debug prints are coming, then that interface is working. User can verify booting of EVK from this interface. Try to login with 'root'.

4.2. GPIO testing

This is for verifying the working of GPIO's in EVk. Follow below steps to test GPIO.

1. Boot the board
2. From the console(in kernel) enter below commands to test GPIO:-
   * To set GPIO direction as output,
   $ test_gpio -n <gpio_number> -o <1-high, 0-low>
   * To set and reset a GPIO
   $ test_gpio -n <gpio_number> -s <1-high, 0-low>

* To set GPIO as input
$ test_gpio -n <gpio_number> -i
* TO set a gpio as interrupt
$ test_gpio -n <gpio_number> -e <value 0-none 1-rising 2-falling 3-both>

3. Verify GPIO state by using below command, or verify using multimeter or CRO.
$ test_gpio -n <gpio_number> -g

Following are the details of gpio:-

Gpio number	Functionality
80	System status led

4.3. I2C interface testing

I2C interface verification is for probing available I2C buses to find connected devices. Follow below steps for testing I2C interface:-

-> To test from u-boot:-

1. Boot the board.
2. Stop the code execution at u-boot by interrupting autoboot from u-boot.
3. give below commands to select i2c bus and to probe selected i2c bus:-
   => i2c dev <bus_number> bus_number can be 0,1,2
   => i2c probe

-> To test from kernel(after complete booting)

1. Boot the board to kernel
2. After complete booting, enter below command to probe i2c devices-
   root@timx8m:~# i2cdetect -y -r <i2c_bus_number>
   Here i2c_bus_number can be 0,1,2
   It outputs a table with the list of detected devices on the specified bus. Each cell in the output table will contain one of the following symbols:-
   * "--". The address was probed but no chip answered.
   * "UU". Probing was skipped, because this address is currently in use by a driver. This strongly suggests that there is a chip at this address.
   * An address number in hexadecimal, e.g. "24". A chip found at this address.

4.4. QSPI testing

Detection of QSPI is verified only in u-boot, as we are accessing QSPI only from u-boot. Try below steps to read from and write to QSPI.

1. Boot the board with serial console
2. Stop the boot at u-boot by interrupting autoboot

3. Check qspi detected or not by giving below commands from u-boot
   => sf probe
4. Try QSPI read and write using below commands:-

   => md 0x80000000 20
   => md 0x90000000 20
   => mw 0x80000000 abcdefab 20
   => md 0x80000000 30
   => sf probe
   SF: Detected n25q512 with page size 256 Bytes, erase size 4 KiB, total 64 MiB
   => sf erase 0x0 0x10000
   => sf write 0x80000000 0x0 20
   => sf read 0x90000000 0x0 20
   => md 0x90000000 30

After write and read, RAM location 0x90000000 will update with QSPI values.

4.5. eMMC testing

Following method we can use to access eMMC. eMMC is acting as one of the boot source.
Use below steps to verify whether eMMC is detected or not-

1. Boot the board completely
2. check dmesg prints to verify eMMC detected or not
   $ dmesg | grep mmcblk2
   Verify below print to confirm the eMMC detection.
   mmcblk2: mmc2:0001 DG4016 14.7 GiB
3. Modify partition table using below command:-
   $ fdisk /dev/mmcblk2
   After executing this command, it will wait for some user input to do activities in eMMC.
   Give following input to alter mmcblk2 device -
   d - To delete partition. Select the partition number which the user wants to delete.
   n - To create new partition. While creating new partition, give partition type, start block and endblock as user inputs.
   p - To show partitions in the device
   w - To write modification made for partition table.
   Use above inputs to create new partition in eMMC and try to mount the same.
4. Mount eMMC partition using below commands:-
   $ mkdir /mnt/partition<partition_number>
   $ mkfs.ext4 /dev/mmcblk2p<partition_number> partition_number can be 1,2,3 ...
   $ mount /dev/mmcblk2p<partition_number> /mnt/partition<partition_number>
5. Then try to copy a file or create a new file in any of the eMMC partition
   $ cp <any_file> /mnt/partition<partition_number>/

or

$ vi /mnt/partition<partition_number>/<file_name>

4.6. MiniPCIe testing

Connect MiniPCIe module to the miniPCIe connector of EVK to verify the interface. Use below commands to verify whether the device detected or not,

$ lspci

If the interface is working then this will give details of connected miniPCIe module.

4.7. WiFi Testing

To verify working of WiFi module, we need to configure the same in to both client mode and AP mode. Following are the steps to configure as **WiFi client**:-

-> To connect to secured network:-

1. Boot the device completely.
2. Change the wpa_supplicant.conf file present in /etc/ with SSID and password of network to which we are trying to connect. Refer section 5.1 for sample wpa_supplicant.conf file. Save the file after making changes.
3. Run below commands to connect to network

   $ sudo killall wpa_supplicant

   $ wpa_supplicant -B -i wlan0 -c /etc/wpa_supplicant.conf -D nl80211

   $ udhcpc -i wlan0

   After executing this, EVK will act as WiFi client and EVK will be assigned with an IP.

-> To connect to open network, use steps-

1. Boot the board.
2. run ifconfig.
3. $ iw dev wlan0 connect <ssid_name>
4. To check the status of wifi link.

   $ iw dev wlan0 link.

Following are the steps to configure WiFi module as **AP mode**:-

1. Boot the board.
2. Create udhcpd.conf file in /etc and add content of udhcpc.conf given in section 5.1
3. Verify hostapd.conf is present in /etc and make sure changes specified for hostapd.conf in section 5.1 are present in /etc/hostapd.conf.

   NOTE:- 'wpa=2' in hostapd.conf makes the wifi closed network.Comment the line to make the wifi as open network.
4. For changing the frequency to 5 GHz, change the hw_mode=a and channel number according to the 802.11ac standards in the hostapd.conf file.

5. Do below steps to set as AP mode
   $ sudo ifconfig wlan0 192.168.27.1 up
   $ killall udhcpd && killall hostapd
   $ hostapd /etc/hostapd.conf & udhcpd /etc/udhcpd.conf
6. Try connecting to the SSID timx8m and password timx8m_test from any client device.
7. Verify hostapd.conf is present in /etc and make sure changes specified for hostapd.conf in section 5.1 are present in /etc/hostapd.conf.

Following are the steps to configure WiFi module as **Dual mode**:-

1. Boot the board.
2. Create wpa_supplicant.conf file ,udhcpd.conf file in /etc and add content of udhcpd.conf as given in section 5.1 with interface name wlan0 changed to wlan1 in hostapd.conf and udhcpd.conf
3. Do below steps to set as Dual mode
   $ sudo killall wpa_supplicant
   $ wpa_supplicant -B -i wlan0 -c /etc/wpa_supplicant.conf -D nl80211
   $ udhcpc -i wlan0
   $ iw dev wlan0 interface add wlan1 type __ap
   $ sudo ifconfig wlan1 192.168.27.1 up
   $ killall udhcpd && killall hostapd
   $ hostapd /etc/hostapd.conf & udhcpd /etc/udhcpd.conf
4. Check ifconfig for the display of wlan0 and wlan1 enumeration.
5. Try connecting to the SSID timx8m and password timx8m_test from any client device.
6. Check the connectivity of both AP and station mode by the following steps:-
   $ ping -I wlan0 google.com
   $ ping -I wlan1 192.168.27.2

Both the ping commands must be passed so that we can verify that both AP and station mode is working concurrently.

4.8. Bluetooth Testing

Bluetooth testing include detection of bluetooth module, establishing connection and file transferring through bluetooth. Following are the steps:-

-> To enable bluetooth module

1. Boot the Board
2. Enable bluetooth by removing reset
   # hciattach /dev/ttymxc0 bcm43xx
   # hciconfig -a

-> To establish connection between bluetooth devices

```
# bluetoothctl
[bluetooth]# power on
[bluetooth]# agent on
[bluetooth]# default-agent
[bluetooth]# discoverable on
[bluetooth]# pairable on
[bluetooth]# scan on
[bluetooth]# pair <MAC_ID>
[bluetooth]# trust <MAC_ID>
[bluetooth]# power off
[bluetooth]# power on
[bluetooth]# connect <MAC_ID>
```

-> To transfer files

```
# export $(dbus-launch)
# /usr/libexec/bluetooth/obexd -r /home/root -a -d -n & obexctl
```

-> From board

```
[obex]# connect 6C:C4:D5:6C:C5:BC
[6C:C4:D5:6C:C5:BC] : send <path to file>
```

-> To board

Initiate connection **from the device** which is paired with the board's bluetooth.Try sending files from connected device to target board. Files will be present in /home/root directory

4.9. Audio testing

To test audio, do below steps after completely booting EVK.

1. Use dmesg prints to verify whether audio codec detected or not,

```
# dmesg | grep sound
```

If the output contains print 'imx-wm8524 sound-wm8524: wm8524-hifi <-> 30030000.sai mapping ok' then audio codec is detected and its driver is up.

2. Use aplay commands to test speaker and headphone-

```
# aplay <audio_file_name.wav>
```

NOTE: Please connect an external amplifier before testing the speaker output.

4.10. USB interface testing

Following are the USB interface test cases,

-> To check USB is detected or not

1. Boot the board completely
2. After complete booting, connect USB device
3. Check any usb connected using lsusb command

lsusb

Check the output of lsusb. If the device details are present, then the USB interface is working.

-> To check USB as host

1. Boot the board completely
2. Connect the USB storage device
3. Create Directory to mount USB
   # mkdir /mnt/usb
4. Mount the device to /mnt/usb by following command
   # mount /dev/sda1 /mnt/usb
5. Perform any data transfer

-> To check USB as device

1. Boot the board completely
2. Connect the board to PC
3. Give the following command
   # umount /run/media/mmcblk2p1
   # modprobe g_mass_storage file=/dev/mmcblk2p1 - emmc will be mounted on PC
4. Perform any data transfer

4.11. Ethernet testing

Following are the steps to verify ethernet interface:-

1. Connect Ethernet Port.
2. Boot the board and enter Serial console.
3. Stop at auto-boot by hitting any key.
4. From u-boot, try to print ethernet address
   => printenv ethaddr
   Shows MAC Address
5. Add MAC Address in user network list.
6. Try execute dhcp command from u-boot to get IP
   => dhcp
   Shows the ip address
7. Boot the board to kernel
8. Verify the ethernet interface by giving below command
   # ifconfig eth0
9. Try to transfer files to and from EVK through ethernet to verify its working.

4.12. PMIC verification(DVFS and low power modes)

If the EVK is perfectly booting, then we can say that the PMIC interface is working fine.

Try below I2C commands to check PMIC is detected or not

```
# i2cdetect -y -r 0
```

If it is detected, then the output will contain 'UU' at the slave address.

4.12.1 DVFS verification

Following are the steps to verify DVFS feature:-

1. Boot the board completely.
2. From the console use either method 1 or method 2 to verify DVS:-

   method 1 :-

   ```
   # echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
   # echo <value_in_KHz> > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
   ```

   method 2 :-

   ```
   # cpufreq-set -f <value_in_KHZ>
   ```

   Measure voltage values at VDD_ARM to verify the feature.
3. To verify DVFS, need to load the CPU. Following methods will load the CPU.
   - Run stress-ng application to load CPU
   - Or use 'sudo dd if=/dev/zero of=/dev/null' command to increase the CPU load.

   Based on the CPU load, cpu frequency will change. The corresponding change in voltage we can measure using multimeter or CRO.

4.12.2 Low power modes

This test is for low power mode validation. Try to pull the system in low power modes by using below steps and verify power on each mode.

1. Boot the board completely.
2. form the console use below commands to pull the system into low power modes:-

   ```
   # echo <mem/freeze> > /sys/power/state
   ```
3. Verify power in each modes ie, mem and freeze.
4. Press on/off button to recover from low power modes.

4.13. Testing of current sensor

From I2C commands, we can identify whether current sensor detected or not. Use below steps to verify current sensors:-

1. Boot the board.
2. Use below i2c command to check whether current sensors detected or not

   ```
   $ i2cdetect -r -y 0
   ```

   Check for device 0x40 , 0x41 and 0x42.
3. Check below nodes are present or not

/sys/bus/i2c/devices/0-0040/of_node/compatible /sys/bus/i2c/devices/0-0041/of_node/compatible
/sys/bus/i2c/devices/0-0042/of_node/compatible
If these nodes are present, then the driver is inserted.

4. Try to read the current sensor values by running 'test_currentsensor' application

4.14. Testing of DSI port

1. Boot the board.
2. Connect the HDMI display to the device via DSI to HDMI converter.
3. After the board boots up you should be able to see the date and time in the display.
4. Run the following steps and you should be able to see a 3D demo rotating in the display:-
   $ cd /opt/imx-gpu-sdk/GLES2/ <RETURN>
   $ ./ModelViewer/ModelViewer_Wayland <RETURN>
5. To exit the demo, use CTRL + C

4.15. Testing of CSI port

1. Boot the board.
2. Connect the camera connector the CSI port and display to the DSI port.
3. Execute the following commands to live stream the camera output to the display:-
   - For 1080p @60 fps -> gst-launch-1.0 v4l2src device=/dev/video0 ! video/x-raw,width=1920,height=1080 ! waylandsink
   - For 1080p @30 fps -> gst-launch-1.0 v4l2src device=/dev/video0 ! video/x-raw,width=1920,height=1080, framerate=30/1 ! waylandsink
   - For 720p @60fps -> gst-launch-1.0 v4l2src device=/dev/video0 ! video/x-raw,width=1280,height=720 ! waylandsink

4.16 Testing of RTC

1. Boot the board completely.
2. Change the date and time of the system using
   date -s "19 APR 2020 11:14:00"
   hwclock -w
3. Press the reset button to reset the board.
4. Check the date and time by using the 'date' command.
5. After the board boots up after pressing the reset button, the board must retain the earlier date and time set by us before pressing the reset button.

5. Appendix

5.1 Contents of configuration files to configure EVK as client and AP

wpa_supplicant.conf

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
network={
        ssid="<SSID>"
        psk="<password>"
}
```

udhcpd.conf

```
start 192.168.27.2
end 192.168.27.254
interface wlan0
opt dns 192.168.x.x
option subnet 255.255.255.0
opt router 192.168.27.1
option domain urorgonizaion.edu
set static ip address to interface wlan0
```

Hostapd.conf parameters are:-

```
interface=wlan0
logger_syslog=-1
logger_syslog_level=2
logger_stdout=-1
logger_stdout_level=2
ctrl_interface=/var/run/hostapd
ctrl_interface_group=0
hw_mode=g
channel=1
```

```
beacon_int=100
dtim_period=2
max_num_sta=255
rts_threshold=-1
fragm_threshold=-1
macaddr_acl=0
auth_algs=3
ignore_broadcast_ssid=0
wmm_enabled=1
eap_server=0
wpa=2
wpa_passphrase=timx8m
```