User's Guide

Rev : 3.0.1_3
Date : 23/07/2020

TIXU_MX6Y User's Guide

VVDN Contact:

POE name : *Talents Titus*
Email : *talents.titus@vvdntech.in*
Mobile : *+91 9746921489*

Revision History:

Date	Rev No.	Description	By
19-Mar-2020	3.0.1_1	Internal Release	VVDN
30-Mar-2020	3.0.1_2	Additions in LCD testing , corrections in I2C ,low power mode & USB testing	VVDN
01-Apr-2020	3.0.1_3	OTG support added	VVDN
23-Jul-2020	3.0.1_3_20200723	DVFS frequency, LCD testing sections have been updated	VVDN

Table of contents

Table of Figures

List of Tables

1 Introduction

This document provides the complete details to work with iMX6ULL EVK powered by TI PMICs TPS65218D0 (hereafter specifying as TIXU_MX6Y). Figure 1 and Figure 2 shows the top view and bottom view of EVK.

Figure 1 : Top View of EVK

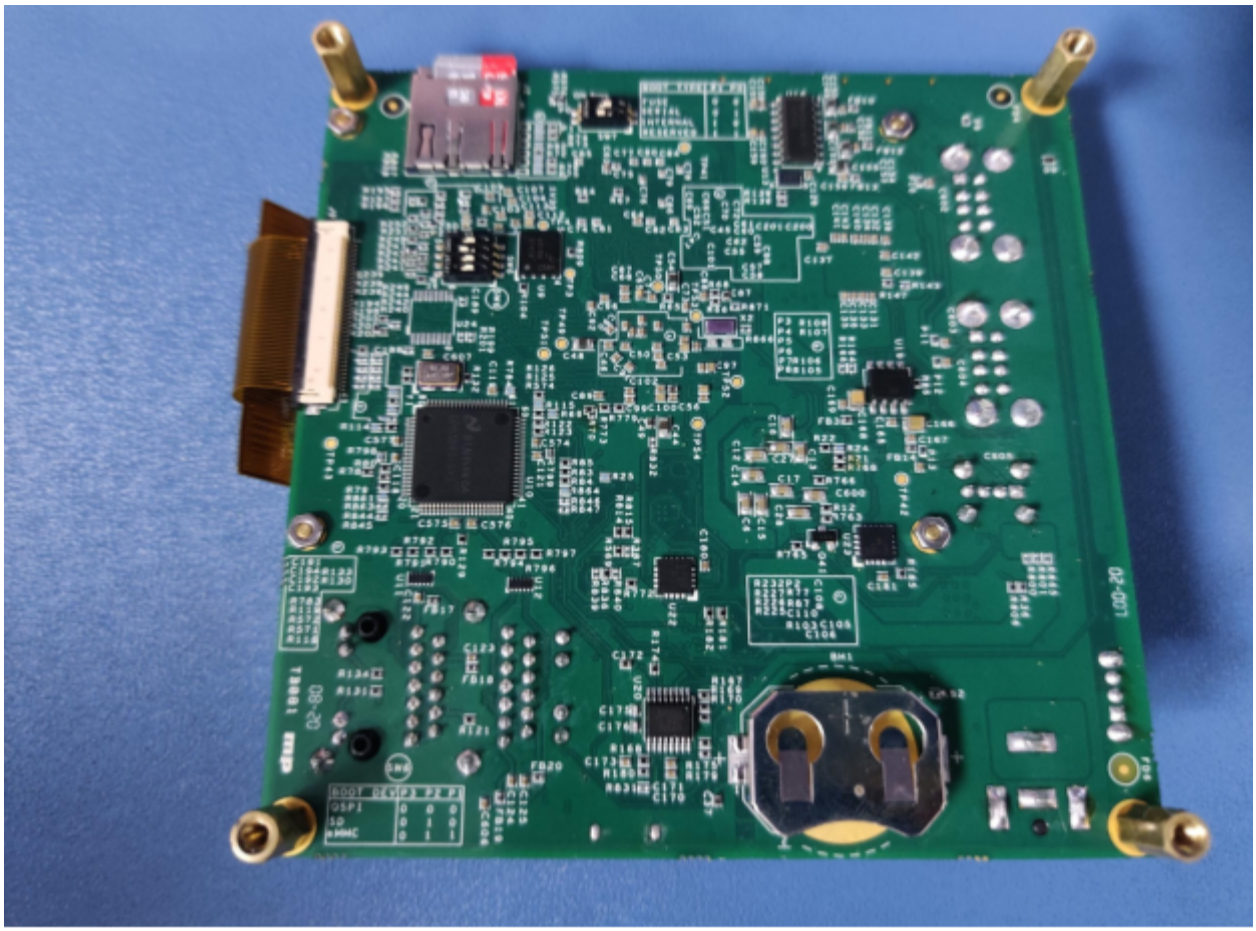

Figure 2 : Bottom View of EVK

2 Preparing SD card for TIXU_MX6Y EVK

The primary boot source we provided for TIXU_MX6Y EVK is an SD card. Use either prebuilt executables or build new executables to prepare SD card. Following section describes the details to prepare SD card either from newly compiled images (after compiling source code from the 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_MX6Y 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 fetch origin
  $ git checkout v3.0.1_2

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

- Use the below command to setup build configurations.
  $ DISTRO=fsl-imx-xwayland MACHINE=timx6y 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_MX6Y to identify changes made by VVDN. So use MACHINE=timx6y in the build configuration to compile sdk for TIXU_MX6Y.
  -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 commands to clean and compile-
  $ bitbake -c cleansstate fsl-image-validation-imx
  $ 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/timx6y/

  Below are the executables(After compilation),

  fsl-image-validation-imx-timx6y-<date_time>.rootfs.sdcard.bz2 - single image containing u-boot, kernel and file system.
  u-boot : u-boot-timx6y.imx
  kernel image : zImage
  Image-timx6y.dtb - kernel device tree
  fsl-image-validation-imx-timx6y-<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-timx6y-<date_time>.rootfs.sdcard.bz2
(replace date_and_time according to compilation time)
$ sudo dd if=fsl-image-validation-imx-timx6y-<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_MX6Y 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_MX6Y 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/timx6y/

  $ bunzip2 -dk -f fsl-image-validation-imx-timx6y-<date_time>.rootfs.sdcard.bz2
  (replace date_time according to compilation time)

  $ sudo dd if=fsl-image-validation-imx-timx6y-<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 represents 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 into SD card.

System requirements are:-

- PC (ubuntu 16.04)
- TIXU_MX6Y EVK
- micro SD card 8GB
- 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 v3.0.1_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 a bootable SD card.
  eg:- *fsl-image-validation-imx-timx6y-20190904153451.rootfs.sdcard.bz2*
- Extract the bz2 image using bunzip2. It will give a sd card image
  $ bunzip2 -dk -f fsl-image-validation-imx-timx6y-<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-timx6y-<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 the SD card is ready for booting TIXU_MX6Y EVK.

2.3 Flashing the eMMC or SD card partition using uuu tool

Refer ***'VVDN_TIXU_MX6Y_WINDOWS_FLASHING_GUIDE_A1'***

3 Booting of TIXU_MX6Y EVK

The primary booting option we provided is an SD card. To flash images into 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.

Use Tera Term 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):

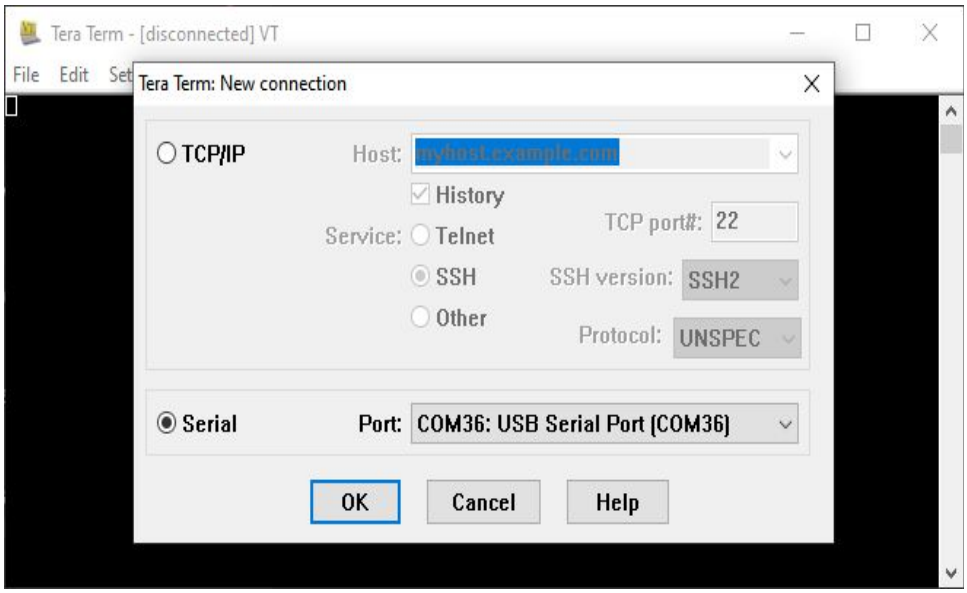

Figure 3 : Opening new Tera Term

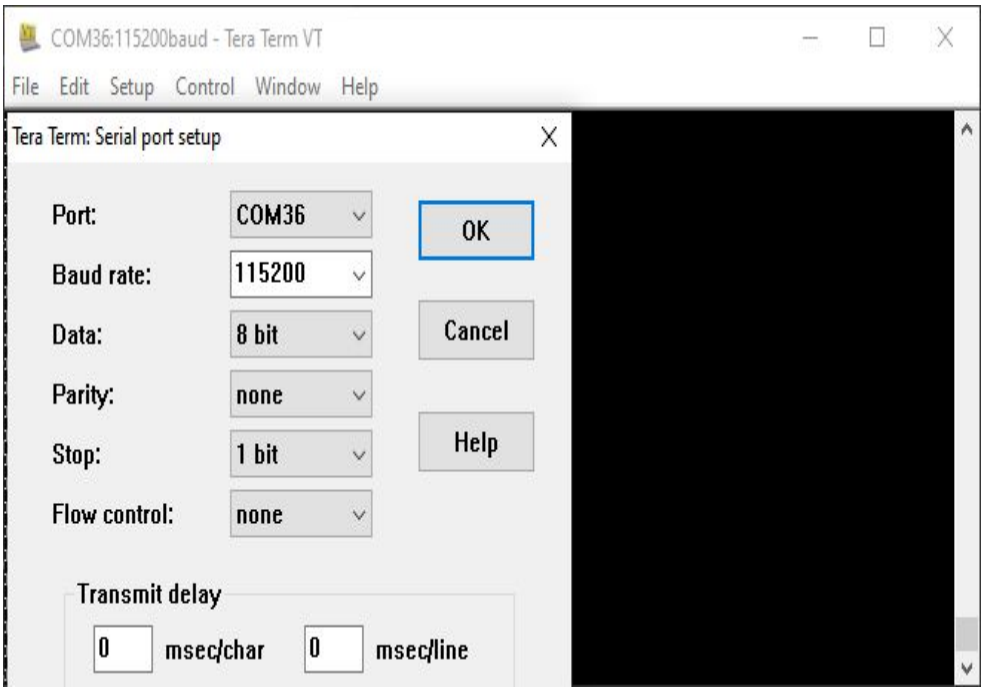

Figure 4 : Tera Term settings

NOTE : Please open the Tera Term or putty after powering on the board since the FTDI chip present in the board is self powered and not bus powered.

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 ttyUSB0> will get displayed
-> Now <Device_Node> is /dev/ttyUSB0 (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. Users can verify booting of EVK from this interface. Try to login with 'root'.

4.1.1 Checking firmware version

1. Boot the board completely
2. Execute the following command to find the firmware version:-

```
# fw-version
```

4.2 GPIO testing

This is for verifying the working of GPIO's in EVK. Follow the steps below 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 / GPIO_Name
115	USR_BT0 / GPIO4_IO19
116	USR_BT1 / GPIO4_IO20
121	USR_LED1 / GPIO4_IO25
122	USR_LED2 / GPIO4_IO26
123	USR_LED3 / GPIO4_IO27
2	REV_CHANGE / GPIO1_IO02
8	HUB_RESET# / GPIO1_IO08
128	PMIC_WAKE# / GPIO5_IO00
129	PMIC_PFO# / GPIO5_IO01
113	PMIC_INT# / GPIO4_IO17
3	ETH_RESET# / GPIO1_IO03
27	TOUCH_BUSY / GPIO1_IO27
26	TOUCH_INTR_REQ# / GPIO1_IO26
23	LCD_PWR_EN / GPIO1_IO23
114	DEBUG_UART_RESET# / GPIO4_IO18
68	LCD_RESET# / GPIO3_IO04
22	GPIO Header / GPIO1_IO22
130	GPIO Header / GPIO5_IO2
131	GPIO Header / GPIO5_IO3

132	GPIO Header / GPIO5_IO4
133	GPIO Header / GPIO5_IO5
134	GPIO Header / GPIO5_IO6
135	GPIO Header / GPIO5_IO7
136	GPIO Header / GPIO5_IO8
137	GPIO Header / GPIO5_IO9
67	GPIO Header / GPIO3_IO3
24	ENET_INT_A / GPIO1_IO24
25	ENET_INT_B / GPIO1_IO25
9	WDOG / GPIO1_IO09

Table 1 : GPIO Pins

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 0,1,3

   => 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@timx6y:~# i2cdetect -y -r <i2c_bus_number>

   Here i2c_bus_number can be 0,1,3

   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 the 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 n25q256a with page size 256 Bytes, erase size 4 KiB, total 32 MiB
   => sf erase 0x0 0x10000
   => sf write 0x80000000 0x0 20
   => sf read 0x90000000 0x0 20
   => md 0x90000000 30

After writing and reading, RAM location 0x90000000 will update with QSPI values.

4.5 eMMC testing

Following methods we can use to access eMMC. eMMC is acting as one of the boot sources. 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 mmcblk1

   Verify below print to confirm the eMMC detection.

   mmcblk1: mmc1:0001 Q2J55L 7.09 GiB
3. Modify partition table using below command:-

   $ fdisk /dev/mmcblk1

   After executing this command, it will wait for some user input to do activities in eMMC.

   Give following input to alter mmcblk1 device -

   d - To delete partition. Select the partition number which the user wants to delete.

n - To create a new partition. While creating a new partition, give partition type, start block and endblock as user inputs.

p - To show partitions in the device

w - To write a modification made for the partition table.

Use the above inputs to create a 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/mmcblk1p<partition_number> partition_number can be 1,2,3 ...

$ mount /dev/mmcblk1p<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 USB Host 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/mmcblk1p2

modprobe g_mass_storage file=/dev/mmcblk1p2 - emmc will be mounted on PC

4. Perform any data transfer

4.7 USB OTG testing

1. Boot the board completely
2. After complete booting, connect the USB device to the board via an OTG cable.
3. Run df command

   # df

   Pendrive will get mounted in /dev/sda1 partition /dev/sda1 -> mounted to -> /media/ubuntu/<Pendrive_name>

   Note: Pendrive is taken as an Example.

4.7.1 File transfer : Sending

1. Send a file from target board to Connected OTG

   # ls > /home/ubuntu/otg_storage.txt

   # cp /home/ubuntu/otg_storage.txt /media/ubuntu/<Pendrive_name>
2. File will get stored in Mass Storage Partition

   # /media/ubuntu/<Pendrive_name>
3. Remove the OTG from target board and connect to laptop
4. Pendrive will be auto-mounted in Laptop

4.7.2 File transfer : Receiving

1. Receive a file from mass storage partition

   * mount mass storage partition on laptop
   * copy files to Mass Storage Partition
   * unmount the pendrive from laptop
2. In Target board, Plug the OTG Cable. Run df command

   # df

   Pendrive will get mounted in /dev/sda1 partition /dev/sda1 -> mounted to -> /media/ubuntu/<Pendrive_name>
3. Check the receive file in /media/ubuntu/<Pendrive_name>/file_name

4.8 Ethernet testing

Following are the steps to verify ethernet interface:-

1. Connect Ethernet Port.
2. Enter the Serial console.
3. Stop at auto-boot by hitting any key.
4. U-Boot >

   setenv ethaddr <MAC_ADDR>

   setenv eth1addr <MAC_ADDR1>

Sets MAC address for both the ethernet interfaces

5. Add Mac Addr. in VVDN Network list (IT Team).
6. U-Boot > dhcp

Shows the ip address

7. Boot the board to kernel
8. Get the ip address by giving the following command.

    udhcpc -i eth0

    udhcpc -i eth1

9. Verify the 2 ethernet interfaces by giving the following command

    ifconfig eth0

    ifconfig eth1

10. Check the connectivity to the internet by giving the following commands

    ping -I eth0 google.com

    ping -I eth1 google.com

11 Successful ping shows both the ethernet interfaces are working correctly.

4.9 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.

4.9.1 DVFS verification

Following are the steps to verify DVFS feature:-

1. Boot the board completely.
2. To measure the DVFS voltage measure the voltage at TP49.
3. 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_CAP to verify the feature.

4. 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 a multimeter or CRO.

Table 2 shows the frequency supported by the processor and corresponding VDD_ARM_CAP voltage.

Frequency in KHz	VDD_ARM_CAP Voltage
792000	1.225V
528000	1.175V
396000	1.025V
198000	0.95V

Table 2 : DVFS Frequency and corresponding Voltage

4.9.2 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/standby> > /sys/power/state
3. Verify power in each mode ie, mem and standby.
4. Press on/off button to recover from low power modes.

- If we want the processor to return to wake up from low power mode when RETURN key is pressed, then the following command needs to be executed before the processor is made to enter into any of the low power modes.

  # echo enabled > /sys/class/tty/ttymxc0/power/wakeup

- We can also make RTC wake up to wake the processor from low power mode by executing the following command before the processor enters any of the low power modes.

  # echo +3 > /sys/class/rtc/rtc0/wakealarm

  This will make the system auto resume 3 seconds after suspend.

4.9.3 Testing of PMIC interrupts

1. Boot the board completely.
2. To test TPS65218 PMIC interrupt, press push button (SW2) which will generate an interrupt as shown below

```
root@timx6y:~# tps65218 0-0024: PB_INT interrupt is set.
tps65218 0-0024: PB_INT interrupt is set.
```

Figure 5 : TPS65218 PMIC interrupt

4.10 LED testing

Below are the test cases to verify working of LED's,

1. Boot the Board
2. After Boot completed execute the test_button_led binary
   # test_button_led -l -s <1-on 0-off> -d <duty cycle from 10 to 100 (multiple of 10))>

4.11 Button testing

Use below test application to verify the working of buttons available in EVK-

1. Boot the Board
2. After Boot completed execute the test_button_led binary

   # test_button_led -b <button_number> -e <event_number>

Button_number can be 1(USR_BT1) to disable LS23 & 2 (USR_BT2) to enable LS23 .

Event_number (set interrupt edge to the pin)can be 0(none), 1(rising), 2(falling) & 3(both).

4.12 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 1
   Check for device 0x40 and 0x41.
3. Check below nodes are present or not

/sys/bus/i2c/devices/1-0040/of_node/compatible
/sys/bus/i2c/devices/1-0041/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.

4.13 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.

4.14 Testing of LCD

LCD can be verified by the following subsections:

4.14.1 Displaying Images & Logo during booting

1. Power on the board.
2. At the starting to till the end of kernel boot up view the display.
3. During boot up view the display for the Tux (Penguin) image and Openembedded image.
4. After we got the login prompt view the display for the TI and VVDN logo.

During the kernel boot up we will be able to see the Tux (Penguin) image and the Openembedded image. Full screen image of TI and VVDN logo will be visible after we get the login prompt in the console. First TI logo will be displayed and after a few seconds VVDN logo will be displayed.

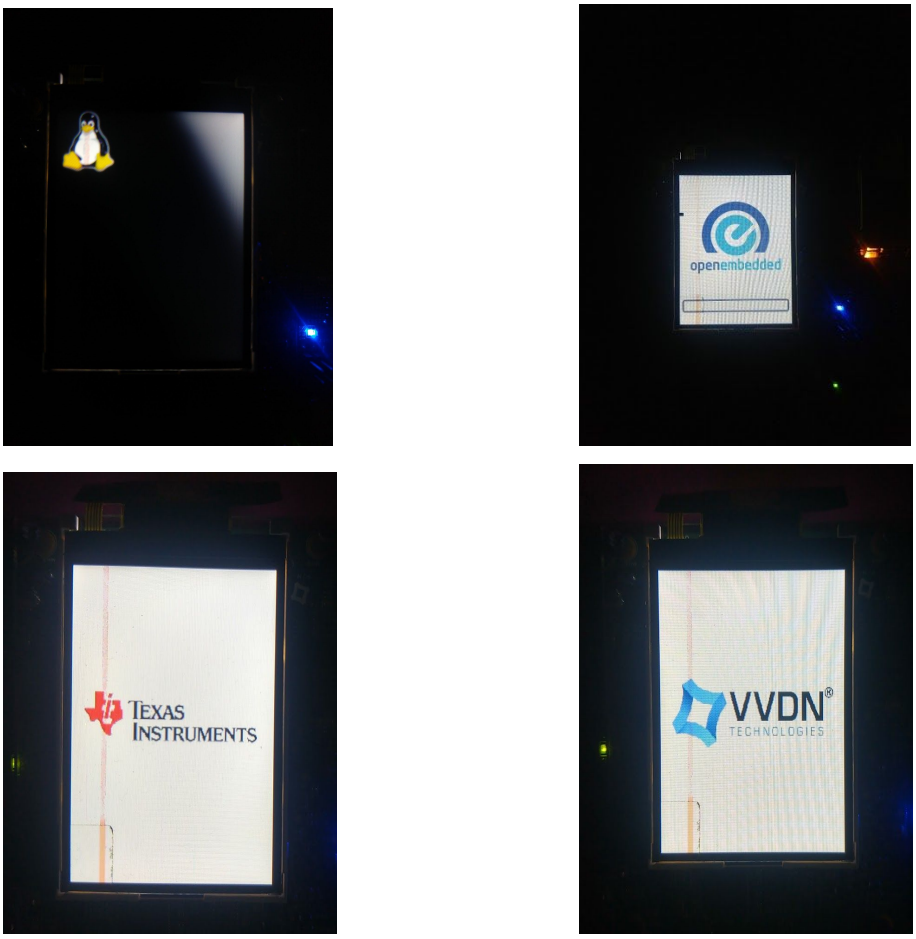

Figure 6 : Images and Logo display

4.14.2 Running application

1.Boot the board completely

2. Run the test application

test_display

Full screen image of Red, Blue, Green stripes would be displayed .

Figure 7 : LCD display showing RGB stripes

4.14.3 Displaying current sensor values

1. Boot the board completely.
2. View the display after complete boot up for the current sensor values in the display.

After the display of TI and VVDN logo the current sensor values will be displayed automatically in the display. First 3 power rail values will be displayed and then after 2 seconds the next 3 power rail values will be displayed and after 2 seconds the first 3 power rail values will be displayed again and this goes on.

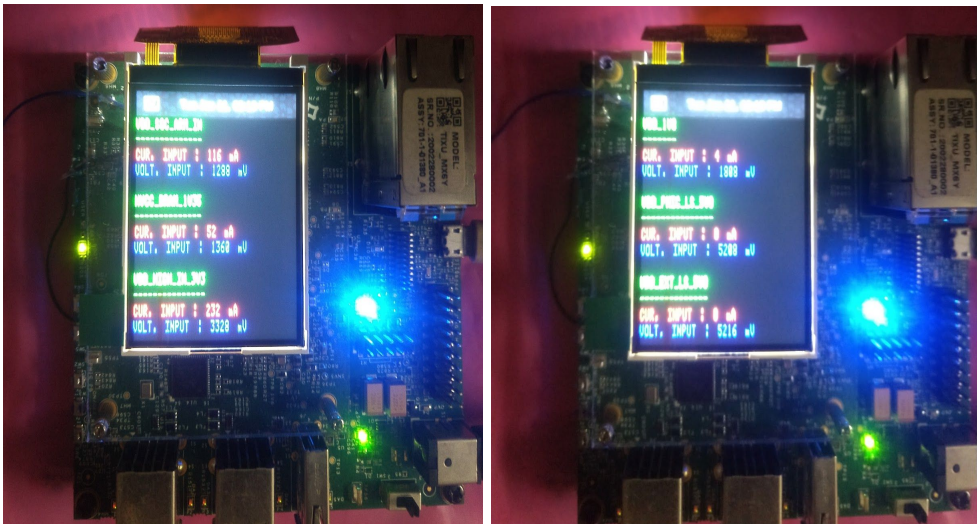

Figure 8 : LCD display showing current sensor values

NOTE: To disable the display of current sensor values, run the following command and the display will be freezed with the last shown current sensor values.

killall -9 current_sensor_display

To resume the display of the current sensor values, run the following command.

```
current_sensor_display
```