PCIe RC-EP communication using TMDS64EVM

Contents

[1. PCIe RC and EP configuration procedure 2](#_Toc176769326)

[1.1. PCIe RC mode configuration 2](#_Toc176769327)

[2. PCIe Endpoint mode configuration 4](#_Toc176769328)

[2.1 Using the Endpoint Test function device 4](#_Toc176769329)

[2.2. Endpoint Controller devices and Function drivers 7](#_Toc176769330)

[2.3. PCIe EPF Test Driver 10](#_Toc176769331)

[3. PCIe RC-EP connecting cable 10](#_Toc176769332)

[3.1 Other vendor of PCIe-Male to PCIe-Male cable: - 11](#_Toc176769333)

[3.2. PCIe 1x cable signal description 13](#_Toc176769334)

[4. Features of AM64x 14](#_Toc176769335)

[4.1. Building kernel for AM64x 15](#_Toc176769336)

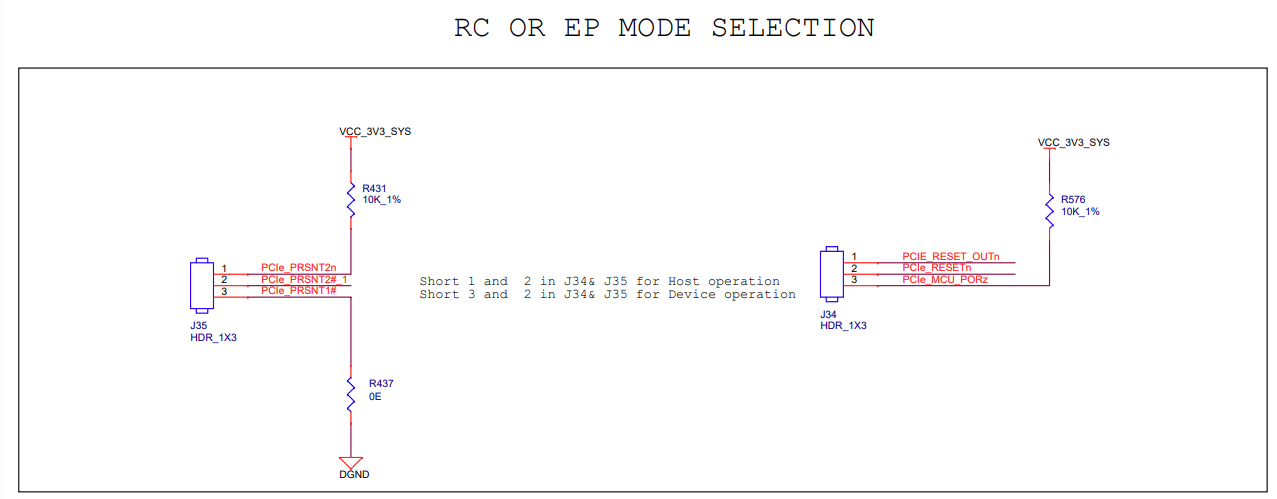
[5. Changing the size of BAR 17](#_Toc176769337)

[References 17](#_Toc176769338)

# PCIe RC and EP configuration procedure

## PCIe RC mode configuration

**Step I. Jumper setting in TMDS64EVM**



Short pin 1 and 2 in J34 and J35 for Root Complex operation.

**Step II. Add below config macros for PCIe RC configuration.**

File: - arch/arm64/configs/ti\_am64x\_pcie\_rc.config

CONFIG\_SOCIONEXT\_SYNQUACER\_PREITS=y

CONFIG\_PCI=y

CONFIG\_PCI\_MSI=y

CONFIG\_PCI\_J721E=y

CONFIG\_PCIE\_CADENCE=y

CONFIG\_PCIE\_CADENCE\_HOST=y

**Change made using “menuconfig”: -**

Device drivers-> PCI support -> PCI controller drivers -> Cdence PCI controller support ->

<\*> TI J721E PCIe platform host controller

**Step III. DTS change**

File: - arch/arm64/boot/dts/ti/ k3-am642-evm.dts

By default, the DTS file – “k3-am642-evm.dts” has configuration as below: -

&pcie0\_rc {

status = "okay";

reset-gpios = <&exp1 5 GPIO\_ACTIVE\_HIGH>;

phys = <&serdes0\_pcie\_link>;

phy-names = "pcie-phy";

num-lanes = <1>;

};

&pcie0\_ep {

phys = <&serdes0\_pcie\_link>;

phy-names = "pcie-phy";

num-lanes = <1>;

};

**PCIe RC driver: -**

“pci\_endpoint\_test”

Driver source code: drivers/misc/pci\_endpoint\_test.c

Config macro used to build this driver: CONFIG\_PCI\_ENDPOINT\_TEST=m

Reference: <linux ksrc>/Documentation/misc-devices/pci-endpoint-test.rst

This misc driver creates /dev/pci-endpoint-test.<num> for every ``pci\_epf\_test`` function connected to the root complex and "ioctls" should be used to perform the above tests.

If TMDS64EVM enumerates “pcie\_epf\_test” EP device, then “pci\_endpoint\_test” driver creates dev node as below.

# ls /dev/pci-endpoint-test.0

/dev/pci-endpoint-test.0

**Step IV. Detect PCIe EP device**

PCIe EP device must be ready and configured as described in next section, as PCIe RC do not support hot plug. After connecting the device, boot PCIE – RC board and run below commands.

echo 1 > /sys/bus/pci/rescan

lspci

# PCIe Endpoint mode configuration

**Step I. Jumper setting in TMDS64EVM**

Short pin 2 and 3 in J34 and J35 for Endpoint operation.

**Step II. Linux driver configuration**

File: - arch/arm64/configs/ti\_am64x\_pcie\_ep.config

CONFIG\_PCI\_ENDPOINT=y

CONFIG\_PCI\_ENDPOINT\_CONFIGFS=y

CONFIG\_PCI\_EPF\_TEST=y

CONFIG\_PCI\_J721E=y

CONFIG\_PCIE\_CADENCE\_EP=y

**Step III. DTS change**

File: - arch/arm64/boot/dts/ti/ k3-am642-evm.dts

&pcie0\_rc { reset-gpios = <&exp1 5 GPIO\_ACTIVE\_HIGH>; phys = <&serdes0\_pcie\_link>;

phy-names = "pcie-phy"; num-lanes = <1>;

status = "disabled";};&pcie0\_ep { phys = <&serdes0\_pcie\_link>;

phy-names = "pcie-phy";

num-lanes = <1>;

status = "okay";

};

## 2.1 Using the Endpoint Test function device

On PCIe Host/RC board, “pci\_endpoint\_test” driver creates the Endpoint Test function device which will be used by “pcitest” utility. “pci\_endpoint\_test” can either be built-in to the kernel or built as a module. For testing legacy interrupt, MSI interrupt has to be disabled in the host.

“pcitest.sh” added in tools/pci/ can be used to run all the default PCI endpoint tests. Before pcitest.sh can be used, pcitest.c should be compiled using following steps:

Cross compilation of pcitest: -

On development PC: -

cd <kernel-dir>

export PATH=/home/mir/My\_Workspace/Cross\_Compiler/arm-gnu-toolchain-11.3.rel1-x86\_64-aarch64-none-linux-gnu/bin:$PATH

make ARCH=arm64 CROSS\_COMPILE=aarch64-none-linux-gnu- headers\_install

aarch64-none-linux-gnu-gcc -Iusr/include tools/pci/pcitest.c -o pcitest

On target console: -

cp pcitest <rootfs>/usr/sbin/

cp tools/pci/pcitest.sh <rootfs>

pcitest output

root@evm:~# ./pcitest -h

usage: -h Print this help message

[options]

Options:

-D <dev> PCI endpoint test device {default: /dev/pci-endpoint-test.0}

-b <bar num> BAR test (bar number between 0..5)

-m <msi num> MSI test (msi number between 1..32)

-x <msix num> MSI-X test (msix number between 1..2048)

-i <irq type> Set IRQ type (0 - Legacy, 1 - MSI, 2 - MSI-X)

-e Clear IRQ

-I Get current IRQ type configured

-l Legacy IRQ test

-r Read buffer test

-w Write buffer test

-c Copy buffer test

-s <size> Size of buffer {default: 100KB}

Sample usage

root@evm:~# ./pcitest -i 1 -D /dev/pci-endpoint-test.0

SET IRQ TYPE TO MSI: OKAY

root@evm:~# ./pcitest -m 1 -D /dev/pci-endpoint-test.0

MSI1: OKAY

root@evm:~# ./pcitest -e -D /dev/pci-endpoint-test.0

CLEAR IRQ: OKAY

root@evm:~# ./pcitest -i 2 -D /dev/pci-endpoint-test.0

SET IRQ TYPE TO MSI-X: OKAY

root@evm:~# ./pcitest -x 1 -D /dev/pci-endpoint-test.0

MSI-X1: OKAY

root@evm:~# ./pcitest -e -D /dev/pci-endpoint-test.0

CLEAR IRQ: OKAY

The script pcitest.sh runs the entire bar tests, interrupt tests, read tests, write tests and copy tests.Output log of “pcitest.sh” script is shown below.

# ./pcitest.sh

BAR tests

BAR0: OKAY

BAR1: OKAY

BAR2: OKAY

BAR3: OKAY

BAR4: OKAY

BAR5: OKAY

Interrupt tests

pci-endpoint-test 0000:01:00.0: Failed to get Legacy interrupt

SET IRQ TYPE TO LEGACY: NOT OKAY

LEGACY IRQ: NOT OKAY

SET IRQ TYPE TO MSI: OKAY

MSI1: OKAY

MSI2: OKAY

MSI3: NOT OKAY

MSI4: NOT OKAY

//….

MSI32: NOT OKAY

SET IRQ TYPE TO MSI-X: OKAY

MSI-X1: OKAY

MSI-X2: OKAY

MSI-X3: NOT OKAY

MSI-X4: NOT OKAY

//….

MSI-X2048: NOT OKAY

Read Tests

SET IRQ TYPE TO MSI: OKAY

READ ( 1 bytes): OKAY

READ ( 1024 bytes): OKAY

READ ( 1025 bytes): OKAY

READ (1024000 bytes): OKAY

READ (1024001 bytes): OKAY

Write Tests

WRITE ( 1 bytes): OKAY

WRITE ( 1024 bytes): OKAY

WRITE ( 1025 bytes): OKAY

WRITE (1024000 bytes): OKAY

WRITE (1024001 bytes): OKAY

Copy Tests

COPY ( 1 bytes): OKAY

COPY ( 1024 bytes): OKAY

COPY ( 1025 bytes): OKAY

COPY (1024000 bytes): OKAY

COPY (1024001 bytes): OKAY

## Endpoint Controller devices and Function drivers

To find the list of endpoint controller devices in the system:

# ls /sys/class/pci\_epc/

f102000.pcie-ep

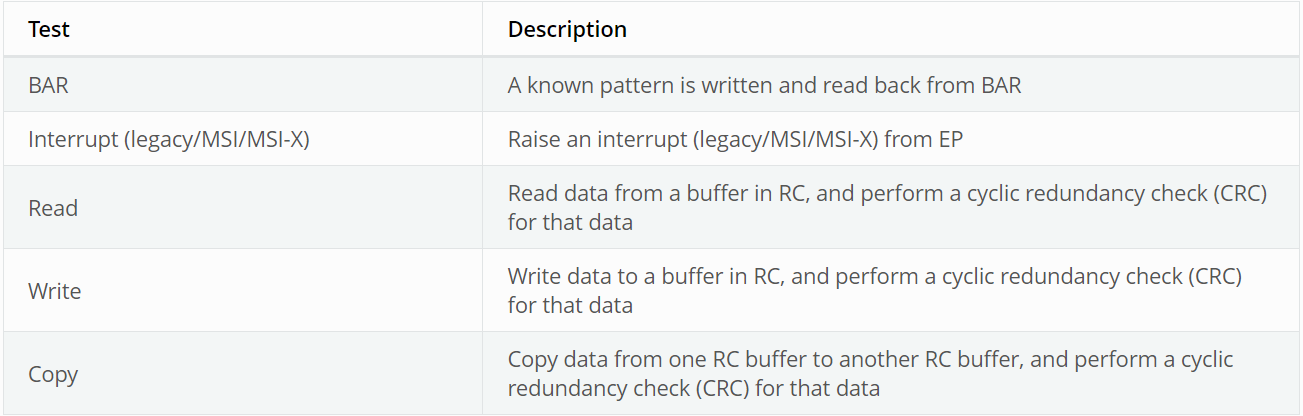
To find the list of endpoint function drivers in the system:

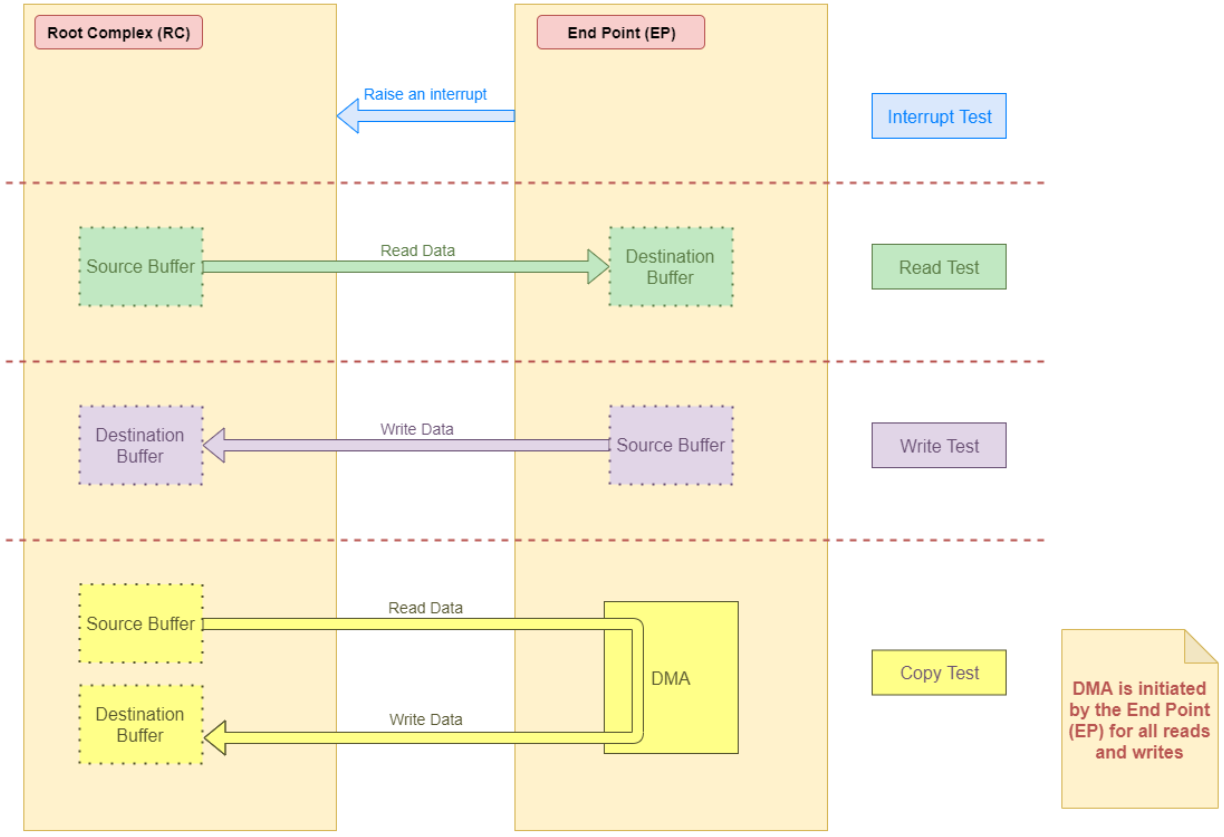
# ls /sys/bus/pci-epf/drivers

pci\_epf\_test pci\_epf\_ntb

**Using the pci-epf-test function driver**

The pci-epf-test function driver can be used to test the endpoint functionality of the PCI controller. Some of the tests that are currently supported are shown below:

****

****

**Creating pci-epf-test device**

PCI endpoint function device can be created using the configfs. To create pci-epf-test function, the following commands can be used:

mount -t configfs none /sys/kernel/config

cd /sys/kernel/config/pci\_ep/

mkdir functions/pci\_epf\_test/func1

The above commands create the pci-epf-test function device.

The PCI endpoint framework populates the directory with configurable fields.

root@evm:/sys/kernel/config/pci\_ep# ls functions/pci\_epf\_test/func1

O/p - baseclass\_code cache\_line\_size deviceid interrupt\_pin msi\_interrupts msix\_interrupts progif\_code revid subclass\_code subsys\_id subsys\_vendor\_id vendorid

The driver populates these entries with default values when the device is bound to the driver. The pci-epf-test driver populates vendorid with 0xffff and interrupt\_pin with 0x0001.

root@evm:/sys/kernel/config/pci\_ep# cat functions/pci\_epf\_test/func1/vendorid

0xffff

root@evm:/sys/kernel/config/pci\_ep# cat functions/pci\_epf\_test/func1/interrupt\_pin

0x0001

**Configuring pci-epf-test device**

1. Configure Vendor ID and Device ID:

(Texas Instruments Vendor ID: 0x104c)

# echo 0x104c > functions/pci\_epf\_test/func1/vendorid

# echo 0xb010 > functions/pci\_epf\_test/func1/deviceid

1. Configure Number of Interrupts

2 is the number of MSI and MSI-X interrupts being configured. The number of interrupts configured should be between 1 to 32 for MSI and 1 to 2048 for MSI-X.

# echo 2 > functions/pci\_epf\_test/func1/msi\_interrupts

# echo 2 > functions/pci\_epf\_test/func1/msix\_interrupts

1. **Binding pci-epf-test device to a EP controller**

In order for the endpoint function device to be useful, it has to be bound to a PCI endpoint controller driver. Use the configfs to bind the function device to one of the controller drivers present in the system.

# ln -s functions/pci\_epf\_test/func1 controllers/f102000.pcie-ep/

1. **Starting the EP device**

In order for the EP device to be ready to establish the link, the following command should be given:

root@evm:/sys/kernel/config/pci\_ep# echo 1 > controllers/f102000.pcie-ep/start

**The complete sequence when using one physical function will look like the following:**

mount -t configfs none /sys/kernel/config

cd /sys/kernel/config/pci\_ep/

mkdir functions/pci\_epf\_test/func1

echo 0x104c > functions/pci\_epf\_test/func1/vendorid

echo 0xb010 > functions/pci\_epf\_test/func1/deviceid

echo 2 > functions/pci\_epf\_test/func1/msi\_interrupts

echo 2 > functions/pci\_epf\_test/func1/msix\_interrupts

ln -s functions/pci\_epf\_test/func1 controllers/f102000.pcie-ep/

echo 1 > controllers/f102000.pcie-ep/start

## PCIe EPF Test Driver

Driver source code: drivers/pci/endpoint/functions/pci-epf-test.c

Config macro : CONFIG\_PCI\_EPF\_TEST

# PCIe RC-EP connecting cable

Cable part# - PE-FLEX1-G2-MMCX-6"-TI1

It is PCIe X1 FLEXIBLE, Male-to-Male, Rx/Tx Cross, G2, 6" Cable, No CLK, PWR Jumpers.

Vendor: Adex Electronics ([https://www.adexelec.com](https://www.adexelec.com/))

Modify the cable to remove resistors in CK+ and CK- in order to avoid ground loops (power) and smoking clock drivers (clk+/-).

 PCIe differential signals Tx to Rx, Rx to Tx, suitable for Master-Slave applications.

Remove the RST resistors to avoid reset (PERST) being propagated from Root Complex to End Point. Also in Root Complex to End Point loopback connection, End Point running Linux should be initialized before Root Complex comes up. Propagating reset from Root Complex to End Point will do PORZ of End Point, which should be avoided.

The ends of the modified cable should look like below:

A Side



B side



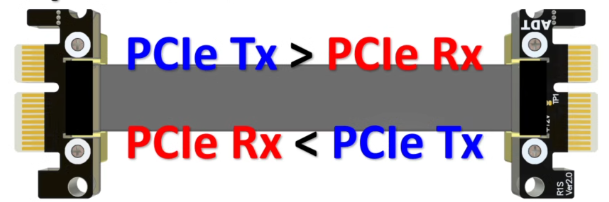
## 3.1 Other vendor of PCIe-Male to PCIe-Male cable: -

Vendor name – ADT-link

https://www.adt.link/product/R11\_Jump.html

Part No. – R11NS

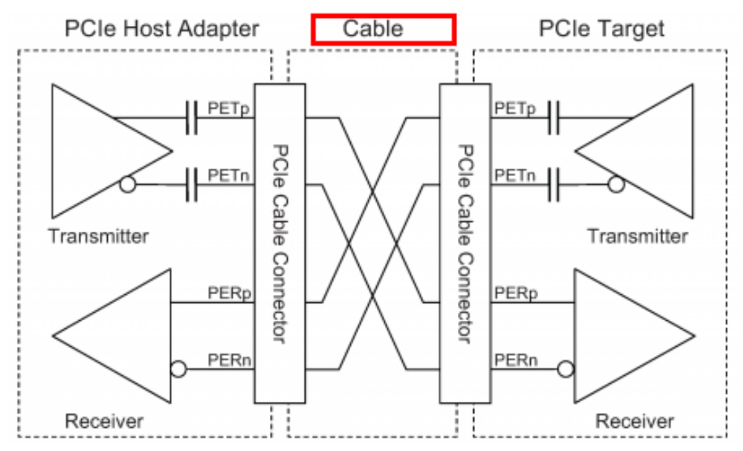




**Notes: -**

<https://wiki.espressobin.net/tiki-index.php?page=Enabling+ESPRESSObin+board+as+PCIe+Endpoint>

This cable **MUST NOT** have clock and power signals connected; otherwise it could damage our board. Consult the image below:



## PCIe 1x cable signal description

PCIe 1x cable has total 18 pins on each side. There are two sides on PCIe connector i.e. Side A and Side B. Hence, there are total 36 pins on a PCIe connector.

In a particular side of PCIe 1x connector, 11 pins and 7 pins groups are separated by key notch.

PCIe signal description on the PCIe connector/cable is shown in below table.

Table – 1.



# Features of AM64x

There is one instance of PCIe subsystem. Following are some of the main features:

* The instance can be configured to operate in Root Complex mode or End Point mode
* One lane configuration, capable up to 5.0 Gbps/lane (Gen2)
* One Physical Function (PF)
* Support for Legacy, MSI and MSI-X Interrupt
* There can be 32 different address mappings in outbound address translation unit. The mappings can be from regions reserved for the PCIe instance.
  + For instance PCIE0, there are two regions in SoC Memory Map:
    - 128 MB region with address in lower 32 bits
    - 4 GB region with address above 32 bits

Note: Source code files and paths

A screenshot of a computer program

Description automatically generated

## Building kernel for AM64x

|  |
| --- |
| Kernel source build guide: -  ------------------------------  Referernce: - https://software-dl.ti.com/processor-sdk-linux/esd/AM64X/08\_05\_00\_21/exports/docs/linux/Foundational\_Components\_Kernel\_Users\_Guide.html  Building kernel: -  -------------------------  ## Go to path of linux kernel in installed SDK –  cd /home/mir/My\_Workspace/TMDS64EVM/TMDS64EVM\_SDK\_Install/SDK\_Install\_Path/board-support/ti-linux-kernel-6.1.46+gitAUTOINC+247b2535b2-g247b2535b2  # Export Tool chain path  export PATH=/home/mir/My\_Workspace/Cross\_Compiler/arm-gnu-toolchain-11.3.rel1-x86\_64-aarch64-none-linux-gnu/bin:$PATH  # make ARCH=arm64 CROSS\_COMPILE=aarch64-none-linux-gnu- distclean  # make ARCH=arm64 CROSS\_COMPILE=aarch64-none-linux-gnu- defconfig  ##For RC  make ARCH=arm64 CROSS\_COMPILE=aarch64-none-linux-gnu- defconfig ti\_am64x\_pcie\_rc.config  ## For EP  make ARCH=arm64 CROSS\_COMPILE=aarch64-none-linux-gnu- defconfig ti\_am64x\_pcie\_ep.config  Or  # make ARCH=arm64 CROSS\_COMPILE=aarch64-none-linux-gnu- tisdk\_am64xx-evm\_defconfig  ## To customize run menuconfig  make ARCH=arm64 CROSS\_COMPILE=aarch64-none-linux-gnu- menuconfig  ## Build Kernel Image  make ARCH=arm64 CROSS\_COMPILE=aarch64-none-linux-gnu- Image  ## Build DTS  # AM64x EVM -> k3-am642-evm.dts  make ARCH=arm64 CROSS\_COMPILE=aarch64-none-linux-gnu- ti/k3-am642-evm.dtb  # make ARCH=arm64 CROSS\_COMPILE=aarch64-none-linux-gnu- dtbs  ## Build Modules  make ARCH=arm64 CROSS\_COMPILE=aarch64-none-linux-gnu- modules  ## Installing the kernel to SD card  cd /home/mir/My\_Workspace/TMDS64EVM/TMDS64EVM\_SDK\_Install/SDK\_Install\_Path/board-support/ti-linux-kernel-6.1.46+gitAUTOINC+247b2535b2-g247b2535b2  mount /dev/sdb2 /mnt  sudo cp arch/arm64/boot/Image /mnt/boot  sudo cp arch/arm64/boot/dts/ti/k3-am642-evm.dtb /mnt/boot/dtb/ti  sudo make ARCH=arm64 INSTALL\_MOD\_PATH=/mnt modules\_install  Note: -  Append INSTALL\_MOD\_STRIP=1 to the make modules\_install command to reduce the size of the resulting installation |

# Changing the size of BAR

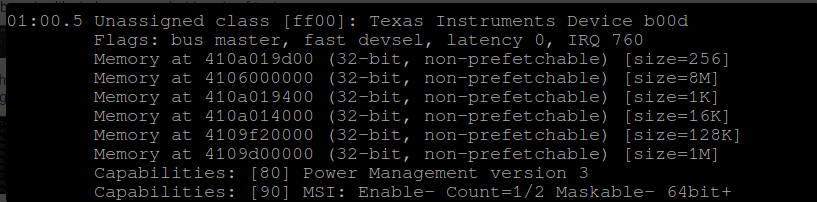
In “pci-epf-test.c”, default size of 6 BAR regions are given as below.

static size\_t bar\_size[] = { 512, 512, 1024, 16384, 131072, 1048576 };

This is just for demonstration purpose, we can even change it, for example 8MB for second bar as below.

static size\_t bar\_size[] = { 512, 8388608, 1024, 16384, 131072, 1048576 };

Now, on lspci output we can see size of BAR as below.



# 

# References

1. <https://software-dl.ti.com/processor-sdk-linux/esd/AM64X/08_06_00_42/exports/docs/linux/Foundational_Components/Kernel/Kernel_Drivers/PCIe/PCIe_End_Point.html#host-device-configuration>
2. https://software-dl.ti.com/processor-sdk-linux/esd/AM64X/09\_00\_00\_03/exports/docs/linux/Foundational\_Components/Kernel/Kernel\_Drivers/PCIe/PCIe\_Root\_Complex.html
3. <https://software-dl.ti.com/processor-sdk-linux/esd/AM64X/09_00_00_03/exports/docs/linux/Foundational_Components/Kernel/Kernel_Drivers/PCIe/PCIe_End_Point.html>
4. <https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1244970/tmds64evm-pci-communication-rc-and-ep-configuration-on-tmds64evm-module-am6442-processor>
5. <https://e2e.ti.com/support/processors-group/processors/f/processors-forum/950450/tda4vm-soc-to-soc-communication-between-two-boards-via-pcie/3518247#3518247>
6. PCIe to M.2 adaptor: -

<https://www.waveshare.com/product/raspberry-pi/boards-kits/compute-module-4-cat/pcie-to-m.2-a.htm>

1. <linux ksrc>/Documentation/misc-devices/pci-endpoint-test.rst
2. <https://en.wikipedia.org/wiki/PCI_Express>
3. <https://elinux.org/Device_Tree_Usage>
4. Increasing the BAR size to 1GB: -

<https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1360433/tda4vh-q1-how-to-modify-pcie-shared-memory-size-to-1gb>

1. https://software-dl.ti.com/mcu-plus-sdk/esd/AM243X/08\_05\_00\_24/exports/docs/api\_guide\_am243x/EXAMPLES\_DRIVERS\_PCIE\_BENCHMARK\_RC.html