Application Note

How to Debug GPU on Jacinto/Sitara SOC

Zekun Bai, Donna Xu

ABSTRACT

The Jacinto™ 7 TDA4 series and Sitara™AM6x processors are the latest generation processors based on the Keystone architecture from TI. The devices integrate different application cores using an on-chip heterogeneous multicore architecture. All those devices have GPU integrated to support 3D rendering accelerate. GPUs have some internal debug tools or trace tools that can report GPU status, error, hardware reset, or any other issues.

Customers usually encountered a GPU driver error reporting many traces in Linux console, or there are also issues manifesting in customer applications. In that situation, how does a customer report this issue to get assistance from TI, and are there any known bugs and patches that customer must use in the GPU driver? This application note discusses those parts and helps to accelerate the debug process.

Table of Contents

Trademarks

1 Basic Debug

1.1 Symptoms

First, describe the symptoms that are shown, for example:

- The display is frozen, went blank, or is flashing.
- There is corruption in the output, such as artifacts or tearing.
- The application *resets* periodically.

1.2 Replication

Provide the following details about the issue:

- How simple is the issue to replicate? What is the time cost and volume of platforms?
- Is this reproducible on a TI EVM? Or is this only available on customer hardware?
- Can the application that is causing the issue be shared?
- Can a demo that can reproduce the issue be shared?

If TI engineers can reproduce the issue, a user can understand the root cause of the issue.

1.3 Recovery

- Can the issue recover itself?
- Can the issue recover by shutting down the application and restarting?
- Can the issue recover by using a rmmod and modprobe GPU driver?
- Can the issue recover by turning off and turning on the device?

1.4 Console Log

Save the console logs when an issue is present. There is a debug system in the GPU driver. When a warning or error is detected, the GPU driver prints the error content or warning content and the call trace. This shows the application running on the GPU and other useful information from the trace, such as the GPU driver version. This is usually is not sufficient and requires further logs.

Here is an example of a console log that reports an PVR error:

```
[ 40.729610] PVR_K:(Error): 253: DevicesWatchdogThread: Device status not OK!!! [500]
[ 40.739858] PVR_K: 253: ------------[ PVR DBG: START (High) ]------------
[ 40.748607] PVR_K: 253: OS kernel info: Linux 5.10.120-g95b90aa828 #1 SMP PREEMPT Sun Apr 23
12:27:10 CST 2023 aarch64
[ 40.762245] PVR_K: 253: DDK info: Rogue_DDK_Linux_WS rogueddk 1.15@6133109 (release) j721e_linux
[ 40.773626] PVR_K: 253: Time now: 40773618us
[ 40.779083] PVR_K: 253: Services State: OK
[ 40.784315] PVR_K: 253: Server Errors: 1
[ 40.789394] PVR_K: 253: Connections Device ID:0(128) P1052-V1052-T1058-byd_srv.out
[ 40.799232] PVR_K: 253: ------[ Driver Info ]------
[ 40.805437] PVR_K: 253: Comparison of UM/KM components: MATCHING
[ 40.813092] PVR_K: 253: KM Arch: 64 Bit
[ 40.818006] PVR_K: 253: UM Connected Clients: 64 Bit
[ 40.824311] PVR_K: 253: UM info: 1.15 @ 6133109 (release) build options: 0x80000810
[ 40.834153] PVR_K: 253: KM info: 1.15 @ 6133109 (release) build options: 0x00000810
[ 40.844315] PVR_K: 253: Window system: wayland
```

2 Further Debug Tracing

2.1 GPU Debug Dump

The services debug dump provides a high-level snapshot of the GPU device and kernel driver state. Note that unless the device or driver is locked up, then the state reported is not consistent. This debug dump can provide logs, but are not enabled by default. To enable and capture these logs, use the following steps:

1. Before running the application, run the following command: *pvrdebug -loggroups main,mts,hwr*.
2. Run the application.
3. After experiencing an error in execution, run the following command: *pvrlogdump*. This generates a file that consists of firmware trace for further analysis.

2.2 Example of Debug Dump

Here is an example of a debug dump from a native driver:

```
------------[ PVR DBG: START (High) ]------------
OS kernel info: Linux 5.10.41-g4c2eade9f7 #1 SMP PREEMPT Fri Apr 29 13:43:19 CST 2022 aarch64
DDK info: Rogue_DDK_Linux_WS rogueddk 1.13@5776728 (release) j721e_linux
Time now: 264880394us
Services State: OK
Connections: P1011-V1011-T1023-tear_test
------[ Driver Info ]------
Comparison of UM/KM components: MATCHING
KM Arch: 64 Bit
UM Connected Clients: 64 Bit
UM info: 1.13 @ 5776728 (release) build options: 0x80000810
KM info: 1.13 @ 5776728 (release) build options: 0x00000810
Window system: wayland
FW info: 1.13 @ 5776728 (release) build options: 0x80000810
------[ RGX Device: Start ]------
------[ RGX Info ]------
RGX BVNC: 22.104.208.318 (rogue)
RGX Device State: Active
RGX Power State: ON
```

OS kernel info:<System Version> <OS Version> <Version Revision> <CPU architecture>

This field indicates the details of the OS the Host is using.

DDK info: <version> (<build type>) <build dir>

This field indicates the PVRVERSION_STRING, PVR_BUILD_TYPE, and PVR_BUILD_DIR. This is a quick identifier for which build target the DDK was built for.

Services State: <state>

This field indicates the state of the driver. Normally set to <OK>.

Server Errors: <error_num>

This indicates the number of FATAL or ERROR debug messages that have been output by the driver since startup.

Device System Power State: <state>

This field indicates the last system power state for the GPU hardware that was notified to the server from the OS layer to the server.

2.3 GPU Status

The service GPU status record GPU behavior since GPU is loaded, such as GPU loading, GPU firmware status, GPU HWR (hardware recovery) event count. Use this command to see the GPU behavior trace, which is helpful for debugging.

/sys/kernel/debug/pvr/status

When the status of normal scenario and abnormal scenario is available, the user can determine if the issue is related to GPU loading or GPU HWR.

Here is an example of GPU status print:

root@j7-evm:~# cat /sys/kernel/debug/pvr/status

Driver Status: OK

Firmware Status: OK

HWR Event Count: 0

CRR Event Count: 0

SLR Event Count: 0

FWF Event Count: 0

APM Event Count: 0

GPU Utilization: 0%

2.4 Disable Auto-Loading of GPU Driver

For debug purposes, the GPU driver can be loaded manually. This is usually done to enable more verbose logging which can only be configured before the GPU driver is loaded. Other times, there can be an issue when the GPU driver is loaded and the driver must be disabled.

The GPU driver is automatically loaded through two initialization scripts. To disable the auto-loading, the following settings have shown to stop the loading of the GPU driver to allow the user to load the driver manually.

1. Make sure that the kernel of the GPU is rejected. Create the file */etc/modprobe.d/blacklist.conf* and add the following to reject the GPU driver: *blacklist pvrsrvkm*
2. Remove the GPU init script. Remove the *rc.pvr file from /etc/init.d*. Move the rc.pvr outside of /etc/init.d so that rc.pvr is not automatically launched. Moving rv.pvr to /home/root is recommended so that the driver can be loaded manually.
3. Load the driver manually

To run the GPU driver manually once the driver has been rejected, run *./rc.pvr start*, which runs the initialization script and load the GPU driver and initialize the driver.

Here is an example to get more information to be enabled in the debugger:

After disable GPU auto loading, add debug parameters when loading the GPU driver. This enables more debug information for analysis.

PVR_SRVKM_PARAMS="EnablePageFaultDebug=1" /etc/init.d/rc.pvr start

pvrdebug -loggroups main,mts,cleanup,csw,bif,pm,rtd,spm,pow,hwr

echo "Y" >/sys/kernel/debug/pvr/apphint/0/AssertOnHWRTrigger

<run application and wait for the Hardware Recovery>

cat /sys/kernel/debug/pvr/firmware_trace > firmware_trace.log

cat /sys/kernel/debug/pvr/debug_dump > debug_dump.log

Finally, obtain the firmware_trace.log and debug_dump.log, and give the log to a TI engineer for analysis.

2.5 GPU Rendering Information

PVRCarbon is a powerful tool that can be used to record and playback an application execution. However, this program takes up a lot of space, especially if the program is running for a long time. This is not feasible if a user cannot sustain storing all of this information until a user sees the issue.

Find more details and the tool on Imagination Technologies' website

2.5.1 Diagnostic

Any issue that can be reproduced from a PVRCarbon capture becomes simpler to investigate. If the issue is reproduced by the PVRCarbon then it is useful to examine the draw calls, textures and Vulkan commands / GLES state to isolate what is causing the problem. Reproducing directly in PVRCarbon render indicates a problem with the application.

If the playback of the trace on a PowerVR test platform reproduces the problem then there is the possibility of a driver problem. If the problem is only reproduced when playing back on the customer device then it is possible there is a hardware related issue.

Here is the usage of record PVRCarbon:

1. Install PVRCarbon onto a Windows or Linux PC with the appropriate installer
2. Copy the contents of the below folder into the target device's filesystem. Use the appropriate architecture, for example, TDA4x devices use Linux_armv8_64/

   PowerVR_Graphics/PowerVR_Tools/PVRCarbon/Recorder/GLES/Linux_armv8_64

   If there is an ssh server running on the target, copy the files over like this:

   scp -r Linux_armv8_64/ root@<ip-addr>:/home/root
3. Use the libraries in the Recorder folder for running the GPU app: export LD_LIBRARY_PATH=<Copied Carbon recorder Location>/Linux_armv8_64
4. Run the application, a user can see PVRCarbon log messages in the console

   PVRCarbon Recorder v0.7 (21.1@5966430)

   PVRCarbon|Note: By default to support multi-platform playback, override api alignment values to our recommended values. To disable this set the align recommended option to false

   PVRCarbon|Writing intermediate data to */run/media/mmcblk1p3/gfxbench5-bins/testfw_app.0001.parts*
5. The PVRCarbon file is created in the directory that the test is ran from. Log examples below:

   - PVRCarbon|Creating file */run/media/mmcblk1p3/gfxbench5-bins/testfw_app.pvrcbn*

   - PVRCarbon|Successfully created */run/media/mmcblk1p3/gfxbench5-bins/testfw_app.pvrcbn*

To play back the PVRCarbon file created, use the following steps:

1. Copy the contents of the below folder into your target device's filesystem:

   PowerVR_Graphics/PowerVR_Tools/PVRCarbon/Player/GLES/Linux_armv8_64
2. Make sure weston is running. If the program is not running, please use the command: /etc/init.d/weston start
3. Copy the PVRCarbon file that was recorded into the Player/GLES/Linux_armv8_64 directory
4. Change directory into the Player/GLES/Linux_armv8_64 on the target device
5. Run: ./PVRCarbonPlayer --ws=wayland *.pvrcbn

2.6 Building a DEBUG Driver

Why build a debug driver?

This is done when an issue is observed on the target device and there is little or no PVR error messages to help diagnose the problem, which is thought to be a GPU related problem. This can also help identify issues in the bring up of a device or issues in the porting of the driver to a different OS. Such issues provide leads into the cause of a problem.

How is this done and what does the driver do?

A DEBUG version of the user-mode driver or the server and kernel-mode driver can be produced by supplying the BUILD=debug macro to the make command or other OS or platform specific build system. On Linux, this compiles the driver code with DEBUG defined. This allows the code to:

- Report warning level messages to the client and server log.
- Enable various debug oriented code in the driver to give more diagnostics.

The application stdout/stderr logs and the Services Server/kernel log contains the additional messages enabled in a DEBUG driver.

Regarding the Kernel mode driver, compile with BUILD=debug.

Because the user-mode driver is not released, contact a TI engineer to help to compile user-mode DEBUG build driver. When the user-mode driver file is obtained, replace the file in the file system and retest.

3 Are there any known issues or patches to fix these issues?

3.1 GPU Cache Coherency Issue

A known bug in the GPU driver for earlier versions is that the driver has a difference cache policy view than the Linux kernel. This affects devices with a MSMC (L3 Cache) and a snoop filter. In some cases, the GPU driver can allocate memory from the kernel with one type of policy but uses the driver in different manners conflicting with the original allocation. The result is random panics or other strange interactions with the kernel.

To mitigate this issue, the following two patches were issued. Both affect the memory cache policy at different levels of the hardware and software stack, and one is more effective than the other. The L3 cache must be disabled as this conflicts with the patches.

3.1.1 GPU Kernel Driver Patch

In SDK 8.5, a kernel driver patch was introduced which covers the majority of GPU memory allocations. The patch is titled *HACK: server: Make CCB allocations incoherent*. This can be found on both SDK 8.5 and SDK 8.6. While this patch fixes the issue for most cases, there are situations where the bug can occur depending on the application implementation. If the issue is on GPU driver versions 1.15, revert this patch and use the following patch which is usable in all cases.

3.1.2 GPU QOS Register Patch: All GPU Transactions

The SoC contains QOS registers that affect a variety of attributes for GPU transactions in the system. One particular register is the *atype* register, which configures the *type* of the GPU transaction. The details are outside of the scope of this document, but the result of setting the atype register to *3* is that the transactions bypass the snoop filter and avoid the issue. The QoS register settings must be applied before the GPU driver is loaded in Linux. There are three alternatives:

1. Apply the a-type value to 3 in the SBL boot flow.
2. Apply the register settings in u-boot of the SPL boot flow. This guarantees the settings is applied before the Linux kernel boots which is before the GPU driver is loaded.
3. Apply the register settings from the Linux command line. This is possible, but requires the GPU driver to not load. This makes the GPU driver to be rejected so that the driver does not auto-start or auto-load.

Use the following instructions on how to implement these three options.

3.1.2.1 SBL Patch

If a user uses the J721E, then QOS setting is enabled by default. So you just need to find this macro and change value to *3* in the SDK and re-compile SBL.

Here is an example of SDK 8.4 of J721E:

File location is ti-processor-sdk-rtos-j721e-evm-08_04_00_06/pdk_jacinto_08_04_00_21/packages/ti/boot/sbl/soc/k3/sbl_qos.c

Change value from 0 to 3.

#define QOS_GPU0_M0_RD_ATYPE (0U)

#define QOS_GPU0_M0_WR_ATYPE (0U)

#define QOS_GPU0_M1_RD_ATYPE (0U)

#define QOS_GPU0_M1_WR_ATYPE (0U)

After modification:

#define QOS_GPU0_M0_RD_ATYPE (3U)

#define QOS_GPU0_M0_WR_ATYPE (3U)

#define QOS_GPU0_M1_RD_ATYPE (3U)

#define QOS_GPU0_M1_WR_ATYPE (3U)

But if a user uses another platform, such as J721S2 or J784S4, enable QOS setting first. Change micro to 3 similar to J721E. See this link to enable this.

3.1.2.2 SPL U-Boot Patch

Apply the following patch to u-boot.

J721E:

0001-HACK-j721e-QoS-workaround-for-GPU-cache-incoherency.patch

Other platforms:

3056.0001-HACK-j721s2-QoS-workaround-for-GPU-cache-incoherency.patch

To verify the patch has been applied successfully, read-back the configuration at any of the modified registers, see that the 0x30000000 has been written into the registers.

3.1.2.3 Linux Command Line Patch

This workaround requires that the GPU driver be rejected so that the GPU driver is not auto-loaded. Please follow the instructions in this FAQ on how to reject the GPU driver and load the driver.

Once you have reached the Linux Kernel console and log in, please run the following script for the respective devices:

J721E

```
#!/bin/bash
devmem2 0x45dc5100 w 0x30000000
devmem2 0x45dc5104 w 0x30000000
devmem2 0x45dc5108 w 0x30000000
devmem2 0x45dc510C w 0x30000000
devmem2 0x45dc5110 w 0x30000000
devmem2 0x45dc5114 w 0x30000000
devmem2 0x45dc5118 w 0x30000000
devmem2 0x45dc511C w 0x30000000
```

J721S2

```
#!/usr/bin/env bash
loops=160
BASE_ADDR_READ=0x45DC5100
BASE_ADDR_WRITE=0x45DC5900
for (( i=0 ; i<$loops ; i++ ));
do
addr=$(($BASE_ADDR_READ + $i * 4))
command='devmem2 ${addr} w 0x30000000'
eval ${command}
addr=$(($BASE_ADDR_WRITE + $i * 4))
command='devmem2 ${addr} w 0x30000000'
eval ${command}
done
```

3.2 Display Tearing or Artifacts

If the display meets these issues and the issues can happen irregularly, use a PHR (Periodical Hardware Reset) to see if this corrects the issue. PHR performs a cache flush.

3.2.1 PHR (Periodical Hardware Reset)

What is PHR?

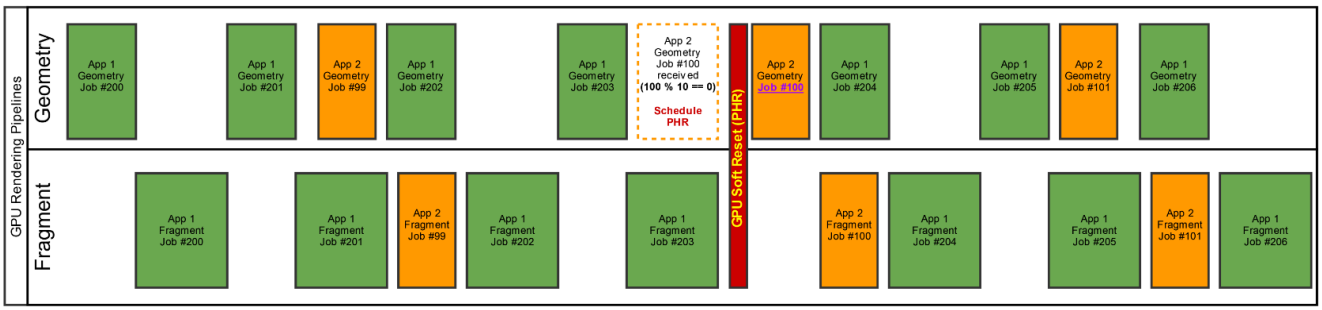
This purpose of the service is to execute cache flush of GPU periodically, once every *n* number of frames of a graphical application are rendered. The resets can be enabled and the frequency can be configured per application, by adding the following entries in /etc/powervr.ini.

Here are examples:

```
 [dashboard_app]
 PHRRate=30
[parking_app]
 PHRRate=10
[navigation_app]
 PHRRate=0
```

If this is configured from the previous steps, the firmware of the GPU performs a graceful reset of the GPU hardware before every 30th frame of the dashboard application, every 10th frame of the parking application and never for the navigation application.

If there are multiple workloads running on the GPU from different applications, when it is time for a Periodic Hardware Reset to be performed for an app, all other currently active workloads are finished, and the HW reset is done before any new workloads are scheduled again.



When you get the user mode driver from TI FAE, you can replace the original with the new set directly and then you can test PHR on board.

How to Enable PHR in My System?

In the default DDK, PHR is inactive. To enable PHR, user mode driver and kernel mode driver must be recompiled. Please contact TI FAE to get access user mode driver. Regarding kernel mode driver, use the following steps:

- Find the file in the Linux SDK, and take J721E as example. The file location is ti-processor-sdk-linux-j7-evm-08_04_00_11/board-support/extra-drivers/ti-img-rogue-driver-1.15.6133109/build/linux/j721e_linux/Makefile
- Add this line in the Makefile: PVR_ENABLE_PHR ：= 1
- Re-compile KM
- Copy the output of KM, pvrsrvkm.ko to the file system.

How to Set the PHR Rate Parameter?

This parameter can be adjusted during runtime in the powervr.ini file. Whenever a graphical app starts, the user space services driver pick up the configuration. So if the value is changed, the targeted apps must be restarted.

Since periodically resetting the GPU slightly affects performance, start with an arbitrary value such as 10 or 30 frames and run some performance testing.

If the performance impact is too high, increase PHR rate gradually, otherwise consider reducing the value to reduce the interval in which the tearing is visible.

How to Make Sure PHR is Enabled When Running the Application?

To check if a reset took place, enable the firmware traces before the reset is expected, then look for the certain logs messages in the firmware_traces. If the GPU is busy with rendering other applications at the same time, note that the Fw message buffer is overwritten quickly, so there is a chance that the PHR traces is not read in time before the traces get overwritten.

Follow the steps to check:

- Enable the firmware traces with

Echo *main* > /sys/kernel/debug/pvr/apphint/0/EnableLogGroup

- Run the application
- Use this command in termininal to grep

cat /sys/kernel/debug/pvr/firmware_trace | grep reset

- If PHR is enabled and application is configured correctly, then a trace such as this is returned:

GPU Hardware units reset to prevent transient faults. The smaller the PHR rate value is, the more traces are obtained.

3.2.2 Lower GPU Frequency or Higher GPU VDD_CORE Voltage

Generally, the higher the working frequency of the GPU, the higher the requirements for external power supply, such as ripple, noise, tolerance and other parameters. When the chip loading jumps, the load current can also jump, causing a voltage drop or glitch on the power supply path. If the layout of the power supply is not very good, the voltage fluctuation can exceed the limit value, causing the GPU module to not work stably. Therefore, TI recommends that customers try to reduce the GPU frequency and increase the GPU power supply voltage.

TI recommends increasing the value by 50-100mV and decrease the GPU frequency by one-third, then look at the actual test results.

How to lower GPU frequency?

K3conf utility can be used to change the GPU frequency. Use the following link to obtain the dev_id and clk_id. k3conf can be used to set the GPU frequency.

Use CTT tools to dump clock registers and find the GPU configuration register. Modify the frequency division coefficient by changing the register value. Use this link for more information about CTT usage.

3.3 Frozen, Blank, or Flashing Display

This issue is caused by HWR (Hardware Reset) usually. If a video is taken of the display, the display turns from normal to dark because HWR is triggered when the GPU has locked up when executing a workload and all ongoing GPU work at the time of the reset is lost. This is a reaction to an error condition that allows the GPU to recover and continue with the next workloads. HWR is different with PHR. When HWR happens, PHR is only triggered by the firmware before periodic geometry workload, while there are no other ongoing workloads on the GPU. No GPU workloads are affected by this soft reset. This is a proactive measure that does not affect the correctness or integrity of any GPU work. As a result, there is no display effect regarding PHR.

But HWR can be triggered by many factors, there are several bugs that must be patched in the file system. Here are related patches of user mode. Contact a TI FAE to obtain and replace GPU libraries.

- Fix_use_of_incorrect_3D_buffers
- Fix_TA_kick_occasionally_being_missed
- Reduce_tiles_in_flight_to_3

4 Summary

This application note summarizes common issues customers encounter when using TI SoC GPU, the symptoms that may occur, and the potential causes behind these symptoms. When encountering similar GPU issues, customers can use the GPU's built-in debug tools to capture more GPU operating status and error information, and submit this information to TI to accelerate analysis and troubleshooting. It also summarizes several common GPU problem countermeasures and patches. Customers who encounter similar issues can try applying these patches to see if the problem is resolved. This will help accelerate the resolution of complex application issues like GPU.

5 References

- Texas Instruments, *TI Live! India Automotive Seminar* , seminar.
- Texas Instruments, *AM62P*, product page.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025