April 2016

NVIDIA Jetson T210 Bringup Automation

Mi Tian
Worcester Polytechnic Institute

Tony Garside
Worcester Polytechnic Institute

NVIDIA Jetson T210 Bringup Automation

A Major Qualifying Project Report:

submitted to the faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by:

_____

Alexander Bragdon

_____

Tony Garside

_____

Mi Tian

Date: March 06, 2016

Approved: _____

Professor David Finkel, Major Advisor

Abstract

The Jetson TX1 development board is NVIDIA's embedded Linux development platform for the Tegra X1 mobile chip. The goal of this project is to automate a series of manual test cases on the Jetson TX1 to verify different on-board modules of the development kit. We accomplished this goal by analyzing existing manual test instructions and automating their execution using Linux Bash script. Internal teams and partner factories of NVIDIA will benefit from these test scripts due to improved test efficiency of the Jetson TX1 development board.

Acknowledgements

We appreciate to get this great chance to Work in NVIDIA. And we would like to thank the following people for their invaluable help and guidance.

Professor David Finkel

Eric Brower

Winnie Hsu

Prabhu Kuttiyam

Sundeep Borra

Mohit Sharma

Shreshtha Sahu

Table of Contents

Abstract **1**

Acknowledgements **2**

Executive Summary ..... Error! Bookmark not defined.

1 Background **6**

1.1 NVIDIA **6**

1.1.1 Main Product 6

1.1.2 Company History 7

1.2 Tegra Chipset **7**

1.2.1 A Brief History 7

1.2.2 Tegra X1 8

2 Test Automation Framework **10**

3 Central Components ..... Error! Bookmark not defined.

3.1 Central Processing Unit (CPU) **11**

3.1.1 Verify Cpufreq Functionality 11

3.1.2 Verify All Cores Online 11

3.1.3 Verify CPU Quiet and Hotplug 11

3.1.4 Verify CPU Frequency 12

3.1.5 Verify CL-DVFS Table 12

3.2 Graphics Processing Unit (GPU) **12**

3.2.1 Verify GPU can run at maximum frequency 12

3.2.2 Verify X11 and Ubuntu desktop are running 13

3.2.3 Verify benchmarks return acceptable results 13

3.3 Thermal Management **13**

3.3.1 Verify CPU throttle 14

3.3.2 Verify GPU throttle 16

3.3.3 Verify thermal zone and cooling devices are registered properly 17

3.3.4 Verify shutdown system on overheat 17

3.4 Fan Pulse-width Modulation **18**

3.4.1 Verify fan speed changes with temperature and can reach maximum RPM . 18

3.5 External Memory Controller 18
3.5.1 Verify EMC DVFS setting is correct 19
3.5.2 Verify EMC is running at maximum frequency 19
3.6 Embedded Multi-Media Controller (eMMC) 19
3.6.1 Verify max eMMC speed meets POR speed 19
3.6.2 Verify read and write of eMMC 20
3.7 Serial Advanced Technology Attachment 20
3.7.1 Verify SATA Online 20
3.7.2 Verify SATA Read Write 20
3.7.3 Verify SATA Resume from Sleep 20
3.8 LP0 21
3.8.1 Verify LP0 Entry and Exit 21
4 Multi-Media Components 22
4.1 Decode 22
4.1.1 Verify gstreamer 22
4.1.2 Verify nvgst-player 22
4.2 Encode 22
4.2.1 Verify Gstreamer GST-OMX encode plugins for H.264 22
4.2.2 Verify Gstreamer GST-OMX encode plugins for VP8 23
4.2.3 Verify Gstreamer GST-JPEG encode plugins 23
4.3 Audio 23
4.3.1 Verify HDMI audio playback 23
4.3.2 Verify headset audio playback 23
4.3.3 Verify headset audio capture 24
4.4 Camera 24
4.4.1 Verify USB camera image and video capture 24
4.4.2 Verify ISP camera image and video capture 24
4.5 High-Definition Multimedia Interface 25
4.5.1 Verify HDMI resolutions 25
4.5.2 Verify EDID can be read 25
4.5.3 Verify CEC is functional 25

5 Peripheral Components 26
5.1 Bluetooth 26
5.1.1 Verify Bluetooth keyboard 26
5.1.2 Verify Bluetooth LP0 exit 26
5.2 Ethernet 26
5.2.1 Verify Ethernet ping 26
5.2.2 Verify Ethernet download 27
5.2.3 Verify Ethernet speed 27
5.3 Peripheral Component Interconnect Express 27
5.3.1 Verify PCIe detected 27
5.3.2 Verify PCIe Ethernet ping 27
5.3.3 Verify PCIe Ethernet functions after suspend/resume 27
5.3.4 Verify PCIe wifi ping 28
5.3.5 Verify PCIe wifi functions after suspend/resume 28
5.4 SD Card Reader 28
5.4.1 Verify SD card detection 28
5.4.2 Verify SD card read and write 29
5.4.3 Verify SD card insertion can wake up system 29
5.4.4 Verify SD card write rejection in write protection mode 29
6 Results 30
7 Conclusion 31
8 Future Work 32
References 33

1 Background

1.1 NVIDIA

NVIDIA, founded in 1993, is a world leader in the visual computing field [1]. The company, headquartered in Santa Clara, California, has more than nine-thousand employees worldwide. Beginning as a standard PC graphics chip company, NVIDIA has lately transformed into a platform company that targets four main markets: gaming, professional visualization, data center and auto. It is also opening up new areas to explore, discover and create technologies such as artificial intelligence and autonomous cars. Currently, NVIDIA holds more than seven-thousand patent assets related to graphic technology Intellectual Property (IP), the largest graphics related holding in the world [2].

1.1.1 Main Product

NVIDIA's products include processors and technologies such as NVIDIA GRID, Quadro VCA, NVIDIA DRIVE, and SHIELD. The following is a list of NVIDIA's main products:

NVS: NVIDIA NVS graphics boards were developed to fulfill the need of driving multiple display devices in business application scenarios, such as display walls used to show stock information in stock trading center or multiple monitors to show airline update in the airport [4].

GeForce: NVIDIA GeForce Graphics Processing Units (GPUs) were designed as discrete GPUs for add-on graphics boards, intended for high-margin PC gaming market. Later variations covered all tiers of PC graphics market, ranging from integrated GPUs on motherboards to mainstream add-in retail boards [4].

Tegra: Tegra is a System-on-a-Chip (SoC) solution for mobile devices. It integrates an ARM architecture Central Processing Unit (CPU) and Graphics Processing Unit (GPU), north-bridge and south-bridge and memory controller onto one package.

Quadro: Quadro was designed for workstations running professional Computer-Aided Design (CAD), Computer-Generated Imagery (CGI) and Digital content creation (DCC) applications. The GPU chips used on Quadro-branded graphics card are identical to the ones

used on GeForce-branded ones. The products are differentiated mainly by the drivers and supporting software.

CUDA: CUDA is a parallel computing platform and programming model designed to increase computing performance by harnessing the power of the GPU.

TESLA: NVIDIA Tesla is designed to target stream processing and/or general purpose GPU to enable accelerated processing necessary for parallelized computing clusters.

1.1.2 Company History

NVIDIA was co-founded by three individuals; Jen-Hsun Huang, previously the Director of CoreWare at LSI Logic and a microprocessor designer at Advanced Micro Devices (AMD), Chris Malachowsky, an electrical engineer who previously worked for Sun Microsystems, and Curtis Priem, previously a graphics chip designer for Sun Microsystems. In 1995, NVIDIA launched its first product, the NV1, a computer graphics card that featured a 2D/3D graphics core made to compete with rivaling 2D graphics cards for IBM PCs. In 1999, NVIDIA invented the product that they are most known for today, the graphics processing unit (GPU), defined by NVIDIA as "a single-chip processor with integrated transform, lighting, triangle setup/clipping and rendering engines that is capable of processing a minimum of 10 million polygons per second." In 2000, NVIDIA collaborated with Microsoft to provide the graphics processors for the Xbox gaming console. In 2005, NVIDIA announced its development of the processor for Sony's PlayStation 3 gaming console. In 2006, NVIDIA revealed CUDA, a revolutionary architecture that enabled software developers/researchers to use the capabilities of parallel processing used by GPUs to solve non-graphics related computational problems [2].

1.2 Tegra Chipset

1.2.1 A Brief History

In 2008, NVIDIA announced its Tegra series of System on a Chip (SoC) platforms to power the coming generations of portable computers. Tegra bundles an ARM CPU with a GPU, and all the controller and devices required in a modern computer, into a single, discreet package. Tegra officially launched to consumers a year later (2009) as a component in the Microsoft Zune HD series of media players [3]. The second generation of Tegra, Tegra 2, was launched a few

months later officially bringing support for Ubuntu, a GNU Linux distribution. Although Tegra 2 was far more widely used for the Android operating system, later versions of Tegra have maintained and greatly improved support for Ubuntu Linux. The next three iterations of Tegra, Tegra 3 through 4i, would continue to increment the platform by increasing performance and lowering power consumption. The sixth generation of Tegra, the Tegra TK1, was subsequently launched in 2014 integrating a five core ARM CPU along with NVIDIA's own Kepler GPU into single chip [5]. Notably the Tegra K1 was the first in the Tegra series to ship on a consumer oriented development board, the Jetson-TK1, running on a version of Ubuntu Linux developed as part of the Linux for Tegra (L4T) program. Both the Jetson development boards and the L4T project would carry forward from Tegra K1 and play a role in 2015 launch of the seventh generation Tegra platform, the Tegra X1.

1.2.2 Tegra X1

Tegra X1 and its accompanying development board, the Jetson-TX1, were launched in 2015 and incorporated the following hardware specifications [6].

- GPU: 1 TFLOP/s 256-core with NVIDIA Maxwell™ Architecture
- CPU: 64-bit ARM® A57 CPUs
- Memory: 4 GB LPDDR4 running at 25.6 GB/s
- Video decode 4K 60 Hz
- Video: encode 4K 30 Hz
- CSI: Up to 6 cameras running at 1400 Mpix/s
- Display: 2x DSI, 1x eDP 1.4, 1x DP 1.2/HDMI
- Connectivity: Connects to 802.11ac Wi-Fi and Bluetooth-enabled devices
- Networking: 1 Gigabit Ethernet PCIE Gen 2 1x1 + 1x4
- Storage: 16 GB eMMC, SDIO, SATA
- Other: 3x UART, 3x SPI, 4x I2C, 4x I2S, GPIOs

When Tegra X1 is shipped to partners and consumers via the Jetson-TX1 module, this full range of features and specifications require assembly line testing to ensure the product is fully functional. However, when partners and consumers integrate Tegra X1 into their products problems may arise between the Tegra X1 and the partner components. Optimally, NVIDIA

would like to distribute these manufacturing tests alongside the Tegra X1 when shipped to partners and clients, so that they can better differentiate between problems on NVIDIA hardware and on the partner hardware. Unfortunately, the testing systems currently used for manufacturing contain too much proprietary technology to be distributed. To alleviate this issue NVIDIA tasked our project team to use the existing testing systems as reference to develop an isolated range of tests that can be distributed without exposing proprietary technology.

2 Test Automation Framework

To accomplish our testing objectives we augmented an existing NVIDIA testing framework designed specifically to run bringup testing. This testing framework, written in the bash scripting language, simply invokes a list of tests given to it based on the particular system the framework is used. Each of the tests, detailed in the following sections, was developed in bash and their filenames added into the framework. When the framework is run, the filenames are used to run the tests, one by one, and the results are recorded and displayed for the user.

3 Core Components

3.1 Central Processing Unit (CPU)

The Central Processing Unit (CPU) is the brain of any computer; its function is to carry out the instructions that enable a computer to compute. On the Tegra X1 the CPU is split into four cores, which allow it to execute multiple tasks in parallel and to manage power through the selective disabling of unused cores. The testing for this module ensures that cores are able to run at max power when needed, but scale themselves back afterword to conserve power.

3.1.1 Verify Cpufreq Functionality

"Cpufreq" is a standard Linux interface for managing and controlling the dynamic scaling of CPU cores and their operating frequencies. To verify that this interface is fully functional the test first configures "cpufreq" to set the first CPU core to its maximum speed, then verifies that the CPU has adopted this new frequency. Next, the test configures the CPU to scale to the lowest speed, and verifies that this speed has been adopted. The test then resets the CPU to its default state. Assuming both of those verifications are true, the test is passed.

3.1.2 Verify All Cores Online

The next test is to verify that all of the CPU cores on the device are operational. To accomplish this test enumerates over all CPU cores in the system using "cpufreq", a linux CPU interface, and sets each of them to explicitly enabled. This prevents the system from powering down cores due to inactivity before verification. Then the test verifies that the system has the correct number of cores using the built-in command "nproc", which returns the number of CPU cores the system currently has. The test then resets the CPU to its default state, and assuming the system has the correct number of active cores, the test is passed.

3.1.3 Verify CPU Quiet and Hotplug

CPU Hotplug is the ability for the system to take CPU cores on and offline without resetting. CPU Quiet utilizes this feature to offline underused cores to conserve power. To verify both of these functionalities the test first enables the CPU Quiet functionality and waits a fraction of a second for the system to adapt to the current usage level, it then verifies that most of the

cores are offline. Then the test disables CPU Quiet and “hotplugs” a particular core offline, and then verifies that only that core has disappeared from the system. The test then resets the CPU to its default state. Assuming the verifications succeeded, the test will pass.

3.1.4 Verify CPU Frequency

The CPU is configured to operate at a specified maximum frequency as specified by the engineers who designed it. This test sets the CPU to operate at this maximum frequency and verifies that the system operates at this exact number. To do this “cpufreq” is used to set the CPU to the maximum frequency and then reads back the operating frequency of the CPU and compares it to the design frequency. Then the system is restored to its default state, and assuming the CPU obtained its maximum frequency the test passes.

3.1.5 Verify CL-DVFS Table

The CL-DVFS (Dynamic Voltage and Frequency Scaling) table is used to define how the CPU should scale it performance based on temperature. To verify the system is complying with the specifications of this table the test overrides the temperature sensor on the CPU and sets it to a range of different temperatures. At each of these temperatures, the CPU must throttle to the specified value from the table. Assuming the CPU throttles correctly for each of the temperatures, the test passes.

3.2 Graphics Processing Unit (GPU)

The Graphics processing unit (GPU) is defined by NVIDIA as “a single chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that is capable of processing a minimum of 10 million polygons per second” [7]. The ever increasing demand for higher quality visuals in digital media has created the constant drive for companies like NVIDIA to design faster, more advanced graphical processing chips.

3.2.1 Verify GPU can run at maximum frequency

The GPU must be capable of running at its maximum frequency according its design specifications. Testing this is simple. The GPU frequency value can be viewed and modified from a specific file that exists in the operating system. To ensure the GPU can run at its

maximum frequency, we set the value in that file to be the GPU's maximum frequency, check that it has in fact changed (if so, the test passes), and finally set it back to what it was before we ran the test.

3.2.2 Verify X11 and Ubuntu desktop are running

When running the Tegra X1 in a graphical context, we must ensure that X11 and Ubuntu desktop are installed and running. X11 is a basic framework for drawing and moving windows to the display. Ubuntu desktop is a software package that includes the core Ubuntu software, including Unity and GNOME, Ubuntu's default desktop environment. It is difficult to ensure that Ubuntu desktop is running. One reliable way of doing so is to check if the 'lightDM' process is running. LightDM is the display manager used by Ubuntu's desktop environment. If lightDM is not running, surely there is a problem with Ubuntu desktop. To accomplish this test, we run simple shell commands to check if X11 and lightDM are running. If these commands return 'true', the tests pass.

3.2.3 Verify benchmarks return acceptable results

It's common practice to use benchmarks to measure a GPU's practical performance. Benchmarks are programs that run computationally expensive calculations to stress computer hardware to measure performance in a standard, repeatable manner. After a benchmark has finished running, the software returns a score which indicates how well the hardware performed against the given computations. To confirm the GPU of the Tegra X1 is meeting performance expectations, we run it against publicly available benchmarks designed to stress the GPU. For now, we ensure 'glmark2' [8] and 'glxgears' [9] benchmarks can run. If they are able to run, then the tests pass.

3.3 Thermal Management

The thermal management module is used for monitoring and managing hardware temperatures of Jetson TX1. Different hardware components belong to different thermal zones and are managed by the thermal framework of temperature sensors and multiple cooling devices. Thermal sensors are built into the Tegra X1 module for components such as the CPU, GPU, memory, etc. Cooling units are used for cooling down the thermal zones, which can be

implemented as hardware devices like fans or as software methods such throttling the CPU or GPU frequencies.

3.3.1 Verify CPU throttle

To verify CPU throttling is correctly functioning, this test needs to verify that when the CPU temperature is increased, the power budget decreases, causing the CPU frequency limit to decrease. To ensure that these changes take place when they should, we simulate changes in the CPU temperature and monitor the internal electrical design point (EDP) variable before and after we make changes to ensure the CPU reacts as we expect. Reducing power budget or the number of online CPU cores can be done by changing the values of system variables. Figure 1 shows the steps to verify CPU EDP management. The first step records the original CPU frequency, which can be compared to later readings after the power budget is reduced, the number of online CPU cores is reduced, and the CPU temperature is increased. As shown in Figure 1, the CPU frequency drops from 1912500 to 1701300 when the power budget is reduced, and increases back to 1912550 after more CPU cores are put offline. In the last step, the frequency drops again to 1874100, due to increased temperature. These numbers meet our expectations because when the power budget is reduced or CPU is temperature increased, the frequency of CPU should reduce to slow down and extend the running time of the computer. However, when fewer cores are online, the CPU frequency of the remaining cores should increase to handle computing tasks from the previously active CPU cores.

1) Check current cpu_edp_limit, max current of vdd_cpu regulator, CPU temperature and number of online CPU cores.

```
# cat /sys/kernel/debug/cpu-tegra/cpu_edp_limit
1912500
# cat /sys/kernel/debug/cpu_edp/reg_edp_ma
25000 ma
# cat /sys/kernel/debug/cpu_edp/temperature
40
# cat /sys/devices/system/cpu/online
0-3
```

2) Reduce the CPU EDP budget and recheck CPU EDP limit

```
# echo 6000 > /sys/kernel/debug/cpu_edp/reg_edp_ma
# cat /sys/kernel/debug/cpu-tegra/cpu_edp_limit
1701300
```

Expected Result: Since vdd_cpu max current is reduced from 25A to 6A, CPU max frequency is also reduced to run CPU within available power budget.

3) Reduce online CPU cores from 0-3 to 0-1 (turn off two cores) and check cpu_edp_limit

```
# echo 0 > /sys/devices/system/cpu/cpuquiet/tegra_cpuquiet/enable
# echo 0 > /sys/devices/system/cpu/cpu3/online
# echo 0 > /sys/devices/system/cpu/cpu2/online
# cat /sys/devices/system/cpu/online
0-1
# cat /sys/kernel/debug/cpu-tegra/cpu_edp_limit
1912500
```

Result: Even though VDD-CPU max is 6A, when the number of online CPUs are reduced from 4 to 2, the CPU frequency limit has been increased due to reduced power consumption from vdd_cpu.

4) Increase CPU temperature and check cpu_edp_limit

```
# echo 1 > /sys/kernel/debug/tegra_soctherm/tempoverride
# echo 100000 > /sys/kernel/debug/tegra_soctherm/cputemp
# cat /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_cur_freq
1874100
#Reset CPU temperature
# echo 0 > /sys/kernel/debug/tegra_soctherm/tempoverride
```

Result: The temperature is directly proportional to power consumption, so frequency max limit has been reduced to run CPU within available power.

Figure 1: The results and steps for verify CPU EDP management

3.3.2 Verify GPU throttle

GPU EDP management is designed to prevent the GPU from drawing more current than its voltage regulator can supply. GPU throttling is tested in a similar manner to how we tested CPU throttling. Figure 2 shows a step-by-step explanation of the verification approach. As shown in the result, the GPU frequency drops from 844800000 to 709600000 hertz after GPU power budget is reduced, and keeps dropping to 691200000 hertz as the GPU temperature increases. The results match our expectations because like in CPU throttling, the GPU frequency is used as a way to react to budget and temperature changes. With lower power budget or higher temperature, the frequency drop will make sure the GPU can run for an extended period of time, at a slower speed.

1) Check current gpu_edp_limit, max current of VDD_GPU regulator, GPU temperature.

```
# cat /sys/kernel/debug/clock/clock_tree | grep edp.gbu
 edp.gbus                 $ on    1        38400000   (844800000^)
# cat /sys/kernel/debug/gpu_edp/imax
25000
# cat /sys/kernel/debug/gpu_edp/temperature
20
```

2) Change the GPU EDP budget and check the GPU EDP limit.

```
# echo 12000 > /sys/kernel/debug/gpu_edp/imax
root@tegra-ubuntu:/home/ubuntu/ppm# cat /sys/kernel/debug/clock/clock_tree | grep edp.gbu
edp.gbus                 $ on    1        38400000   (729600000^)
```

Expected result: since the GPU max current is reduced from 25A to 12A, the GPU max frequency is also reduced from 844800000 to 729600000.

3) Increase CPU and GPU temperature, verify GPU limit is reduced

```
# echo 1 >/sys/kernel/debug/clock/override.gbus/state
# echo 998400000 > /sys/kernel/debug/clock/override.gbus/rate
# echo 0 >/sys/kernel/debug/clock/override.gbus/state
```

Set GPU and CPU tempratures back

```
# echo 1 >/sys/kernel/debug/tegra_soctherm/tempoverride
# echo 100000 > /sys/kernel/debug/tegra_soctherm/cputemp
# echo 100000 > /sys/kernel/debug/tegra_soctherm/gputemp
# cat /sys/kernel/debug/clock/gbus/rate
edp.gbus                $ on    1        691200000 (691200000^)
```

Expected result: The GPU frequency EDP limit should be reduced if GPU temperature is increased.

Figure 2: The steps for verifying GPU EDP management

3.3.3 Verify thermal zone and cooling devices are registered properly

The Jetson TX1 has multiple thermal zones and cooling components which need to be registered properly in the Linux system. This test is implemented to do this verification, by reading and checking that the system directories exist for all expected thermal zones and cooling devices. Specifically, the verified TX1 and external thermal sensors include AO-therm, CPU-therm, GPU-therm, PLL-therm, PMIC-Die, Tdiode_tegra, Tboard_tegra and thermal-fan. They are iterated using the following command:

ls /sys/class/thermal | grep thermal_zone

For each returned path, the test script tries to match it with a known thermal zone. If a match is missing for thermal zone, failure will be reported. Similar approach is used to make sure all cooling devices are registered properly.

3.3.4 Verify shutdown system on overheat

To protect the hardware from overheating and damaging itself, the system should be able to shutdown automatically when the cooling device fails and the temperature climbs above a certain point. There exists a file in the operating system that stores the absolute maximum temperature the TX1 is allowed to reach before causing major damage to itself. One way we could test that this feature is functional is to heat up the TX1 until it reaches this trip point and then observe whether or not it shuts down. This however, is time consuming and risks damaging the TX1's hardware, so instead, we just trick the TX1 into thinking it's overheating when it

really isn't. Our test script modifies the CPU shutdown trip point temperature to a much lower value. Preferably, the value should be even lower than the current CPU temperature, but above typical start up temperature, so that a shutdown can be triggered, but without constantly rebooting the board afterwards. When this test is run, we expect the TX1 to shut down immediately. As of right now, this test does not currently run in our test suite because our testing framework does not have support for resuming after being rebooted.

3.4 Fan Pulse-width Modulation

As it's running, the Tegra X1 module is expected to get quite hot in temperature. To keep the temperature of the chips at a safe level, customers will likely install a heat sink/fan cooler to their module. The following test is designed to ensure the Fan pulse-width modulation is functional.

3.4.1 Verify fan speed changes with temperature and can reach maximum RPM

We expect the revolutions per minute (RPM) of the cooling fan to increase as the Tegra X1 module gets hotter. It would be quite difficult and time consuming to actually heat up the Tegra X1 module every time this test is run. Luckily, we can simulate a temperature increase in the software instead. In this test we trick the module into thinking that its temperature is increasing by overriding the temperature file in the operating system. As we simulate a temperature increase we observe the RPM of the fan via a file in the operating system. If the fan's RPM increases as the temperature increases and it is able to reach its maximum RPM, then the test passes, otherwise it fails.

3.5 External Memory Controller

The external memory controller is the Random Access Memory (RAM) chip on the Tegra X1 module. This is volatile storage for program data and instructions as the module is running. The Tegra X1 comes with 4 Gigabytes of low power DDR4 RAM.

3.5.1 Verify EMC DVFS setting is correct

This test is to ensure that the Tegra X1's RAM can run at a specific range of frequencies. These frequency levels are outlined in the specifications for the dynamic voltage and frequency scaling (DVFS) documentation. To accomplish this test, we compare the values of the possible rates that the RAM can run at to the DVFS specifications. The possible rates that the RAM can run at are located in a file in the operating system, so this test is as simple as comparing the values in that file to the specifications and pass the test if they match.

3.5.2 Verify EMC is running at maximum frequency

This test is to ensure that the Tegra X1's RAM is running at the highest frequency possible. This is as simple as checking the file that stores the current RAM frequency and comparing it to the maximum possible frequency and if they are the same then the test passes.

3.6 Embedded Multi-Media Controller (eMMC)

The Embedded Multi-Media Controller (eMMC) consists of both flash memory and a flash memory controller. eMMC is an onboard chip responsible for data storage. It is slower and cheaper than a traditional Solid State Drive (SSD), and is commonly used inexpensive tablets and laptops. The Jetson TX1 comes with 16-Gigabytes of eMMC flash memory [10].

3.6.1 Verify max eMMC speed meets POR speed

This test is needed to ensure that the eMMC frequency can match the TX1's Plan of Record (POR) frequency. To do this, the test script first queries the POR frequency using the following command:

cat /sys/kernel/debug/clock/clock_tree | grep sdmmc4

This command prints out all of the relevant information that we need to know about the eMMC's clock frequency. The POR frequency is embedded in this output and marked as "sdmmc4". We compare the frequency output by this command with the expected value according to the TX1's specifications and if the two are equivalent, the test will pass.

3.6.2 Verify read and write of eMMC

As the main module responsible for data storage, the eMMC needs to be tested to ensure it can correctly perform read and write operations. To do this, the test script uses an interactive approach. The user is guided to connect an external computer to their TX1 board, and copy a large file from the PC to the board. Once the copy is complete, the same file is copied back to the PC, and saved with a different file name. Lastly, the test script compares the MD5 checksum of the files to determine if they are identical. If the two files are not identical, the test will fail indicating an error has occurred during the read or write process.

3.7 Serial Advanced Technology Attachment

Serial Advanced Technology Attachment (SATA) is a common expansion port that allows users to connect external hard drives for data storage.

3.7.1 Verify SATA Online

This test ensures that SATA devices are detected by the TX1 when connected before its power on. To perform this check, the boot logs of the device are searched for a message indicating a SATA device has been detected. Then the partition tables of the system are analyzed to ensure a new SATA device is present. Assuming both checks are true, the test passes.

3.7.2 Verify SATA Read Write

To ensure the SATA device can properly mount to the system and data accessed from the device a number of tests must be run to ensure proper performance. First the drive is mounted and a small file and a large file written out to it. Then those two files are read back and their contents verified. Assuming that what went out is what came back in, then the test passes.

3.7.3 Verify SATA Resume from Sleep

When coming out of a sleep state it’s a common bug for SATA devices to lose connection. The purpose of this test is to ensure that when recovering from a suspended state that the SATA device can still be written to a read from. This test puts the device into sleep mode, wakes itself, and then tries to performs the same read and write tests from before. If this same test passes again, the test passes.

3.8 LP0

Tegra X1 supports a low power mode called LP0 (Low Power 0) which is a sleep mode that allows the system to draw very little power when not being used. The system must be able to enter and exit this sleep state quickly and without error.

3.8.1 Verify LP0 Entry and Exit

To verify that the system can properly enter and exit LP0 a test is conducted where the number of times the system has slept since last reboot is recorded. Then a wakeup timer is set in the power control unit that will send a wakeup signal to the board ten seconds later. The system is then set to enter LP0 immediacy. Assuming the board successfully enters LP0, and then subsequently exits ten seconds later, the number of times the system will have slept should increase by exactly once. If it has, then the test passes.

4 Multimedia Components

4.1 Decode

The Tegra X1 supports a number of different video and image formats for hardware decoding and playback, which is significantly more efficient than software based decoding methods. To test this feature each of the following tests playback a file for the user and prompts them for feedback on the performance. If the user feedback is positive, then the test passes.

4.1.1 Verify gstreamer

Gstreamer is an open source media playback framework that is included with the Jetson TX1. It is used to playback five different file formats; h264, VP8, MPEG4, MPEG2 and JPEG. Each of the files types is played and evaluated as described above.

4.1.2 Verify nvgst-player

Nvgst-player is an NVIDIA fork of "gst-player" that has been modified to utilize NVIDIA's hardware decoding and is included with the Jetson TX1. Nvgst-payer is used to playback five different file formats; h263, h264, MPEG2, MPEG4 and VP8. Each of the files types is played and evaluated as described above.

4.2 Encode

Video and image encoding is the process of taking the raw data produced by hardware cameras and converting it into the common digital formats that computer applications can process (formats like MP4 or JPEG for example).

4.2.1 Verify Gstreamer GST-OMX encode plugins for H.264

To test that the plugins for H.264 encoding work, our tests run a Gstreamer command that encodes a sample video file into H.264 format. After the video is encoded, it gets played back and the user is prompted to inspect it for any distortion. The sample video is a short recording of SMPTE color bars, which is a common test pattern used to test video. If the user finds no glitches or distortion, the test passes.

4.2.2 Verify Gstreamer GST-OMX encode plugins for VP8

To test that the plugins for VP8 encoding work, we do almost exactly the same as we did for H.264 plugins. The test runs a Gstreamer command that encodes a sample video file into VP8 format. We use the same sample video as we did with H.264, and again, as have the user input whether or not they see any distortion. If they find no problems with the quality of the test video, the test passes.

4.2.3 Verify Gstreamer GST-JPEG encode plugins

To test that JPEG encode plugins are working properly, we run a command that takes a picture using the default camera detected by the application, encodes it into a JPEG image, and displays it to the user. The user is then prompted to inspect the image and input whether or not the image looks acceptable.

4.3 Audio

The Jetson TX1 supports audio playback for HDMI and headset devices. In addition to playback, connected headset devices should also be capable of recording as well. Due to restrictions that disallow us to use third-party software, in order to ensure that audio can play without distortion, these tests need to be interactive.

4.3.1 Verify HDMI audio playback

To test HDMI audio playback, we first set the default audio playback device to be whatever is hooked up to the HDMI port. Then we playback audio in MP3, AAC, and WAV formats using a software called APLAY. APLAY is a command line sound file player. Finally, we display a prompt asking the user if the audio actually played back or not through their HDMI device. If the user responses ‘yes’, the tests pass, otherwise they fail.

4.3.2 Verify headset audio playback

To test Headset audio playback, we do essentially the same thing we did for our HDMI audio tests. We detect the headset device that is connected to the TX1 and set it to be the default audio playback device. Then we playback audio in MP3, AAC, and WAC formats using APLAY and prompt the user to tell us whether or not they heard audio playback through their headset.

4.3.3 Verify headset audio capture

To test headset audio capture, first, we detect the headset device that is connected to the TX1 and set it to be the default audio capture device. We prompt the user to speak into the microphone of their headset as we run a piece of software called ARECORD. ARECORD is a command line sound file recorder. We record 5 seconds of audio and then play it back to the user. At this stage of the test, the user may repeat the recording step multiple times if they are unsure of whether their recorded audio is playing back. Finally, we give the user a prompt asking whether or not they were able to record audio, if 'yes' then the test passes, otherwise it fails.

4.4 Camera

The Tegra X1 supports up to 6 CSI cameras recording at 1400 megapixels per second. Additionally, the X1 supports USB cameras as well.

4.4.1 Verify USB camera image and video capture

To ensure USB cameras are functional, we need to make sure they can capture still images and video without any distortion. These tests are partially automated, but still require users to inspect the captures for corruption and distortion. To test image capture, we use NVIDIA's built in image capture software to take a picture using the default USB camera that is connected to the module. Next we show the user the image that was just captured and give them a prompt asking them if the image looks acceptable, if they respond 'yes' to the prompt then the test passes, otherwise it fails. Unfortunately, it is not possible at this time to automate the testing of video capture with USB cameras due to software restrictions. To work around this, we document instructions to test video capture manually. The instructions walk the user through taking a video capture using NVIDIA's built in video recording software. After the user takes a recording they are instructed on how to play their recording back to ensure that the recording worked properly.

4.4.2 Verify ISP camera image and video capture

Testing to ensure ISP cameras are functional is very similar to how we tested USB cameras. For still image capture we run a command to take a picture using the default ISP camera connected to the module. Next the program opens the image and prompts the user to

inspect the image. Unlike USB camera video capture, ISP video capture can be partially automated. The test program executes built in video recording software to take about 5 seconds of video. The video is then played back for the user and they are asked whether the video recording quality was acceptable, if 'yes', the test passes.

4.5 High-Definition Multimedia Interface

The Tegra X1 supports a single High-Definition Multimedia Interface (HDMI) port to allow NVIDIA's customers to develop projects the utilize an HDMI display. HDMI is a proprietary interface for transferring audio and video data from a source device (in our case the Tegra X1) to a host device (in most cases being a display of some sort) [10].

4.5.1 Verify HDMI resolutions

This test verifies that when an HDMI display is connected to the Tegra X1, that it can render properly at its specified supported resolutions. The display resolution is the number of pixels being displayed on the screen length by width. We test that images and text can be displayed properly at the following resolutions: 600x480, 720x576, 1280x720, and 1920x1080. The test runs a command to change the display resolution and then asks the user whether or not images and text are being rendered properly.

4.5.2 Verify EDID can be read

This test verifies that the Extended Display Identification Data (EDID) can be read. EDID is a data structure that contains the capabilities of a video display. This test is run to ensure that the display is being read properly. To do this, the test runs a command to print out the EDID of the display currently connected to the TX1. If data is returned, the test passes.

4.5.3 Verify CEC is functional

Consumer Electronics Control (CEC) is a feature that HDMI supports to allow users to control CEC-enabled devices using a single remote. This feature allows devices such as DVD players and audio systems to all be controlled with one Television remote. To test this feature, we use a specific program used to connect CEC devices. We run a specific command to identify if any CEC devices are detected, if yes, the test passes.

5 Peripheral Components

5.1 Bluetooth

Bluetooth wireless communication functionality is built into the Tegra X1 and is commonly used for connecting various Human Interaction Devices (HID) such as keyboard and mice.

5.1.1 Verify Bluetooth keyboard

This test automates the process of pairing and connecting a Bluetooth keyboard using the BlueZ Bluetooth stack, which is included with the Jetson TX1. Once the keyboard has been connected, the test is passed if a new keyboard device has been registered by the system.

5.1.2 Verify Bluetooth LP0 exit

This test verifies that Bluetooth HID devices can force the system out of a LP0 sleep state. To perform this test, the user is asked to connect a Bluetooth keyboard or a mouse, or to exit the test. If a Bluetooth device is connected the device enters LP0 sleep and waits for the user to wake it. Once resumed the test is passed if the user verifies that they were able to exit sleep using the Bluetooth device, and not another method.

5.2 Ethernet

Ethernet is a computer networking technology used to connect computer devices to the internet.

5.2.1 Verify Ethernet ping

This test ensures that the onboard Ethernet can connect to the internet. The test accomplishes this by pinging NVIDIA's website using the default Ethernet connection. If it can successfully ping NVIDIA's website, the test passes.

5.2.2 Verify Ethernet download

This test ensures that the TX1 can download files via Ethernet connection. This test is accomplished by running a command to download the index page of NVIDIA's website. If the index page is successfully downloaded using the TX1's Ethernet connection, the test passes.

5.2.3 Verify Ethernet speed

This test ensures that the speed of the Ethernet connection is detected properly. The TX1's Ethernet speed should be 1Gb/s. To check the speed of the Ethernet connection we can run an Ethernet tool command. The command can return the current speed of the Ethernet connection, if the current speed is 1Gb/s, the test passes.

5.3 Peripheral Component Interconnect Express

Peripheral Component Interconnect Express (PCIe) is a common expansion port that allows users to connect additional devices such as an additional wireless card or a dedicated graphics card to their TX1. This may be done for a variety of reasons, perhaps for performance benefits or additional functionality.

5.3.1 Verify PCIe detected

This test ensures that a PCIe device is detected by the TX1. This test is useful for basic troubleshooting purposes. The script runs a command to list all of the PCI devices connected to the board. If it doesn't find any devices, the test fails.

5.3.2 Verify PCIe Ethernet ping

This test is used to test internet connectivity with a PCIe Ethernet card. If it can successfully send a ping to NVIDIA's website using the Ethernet port of the connected PCIe device, it passes.

5.3.3 Verify PCIe Ethernet functions after suspend/resume

When coming out of a sleep state, it is a common bug for expansion devices to lose functionality. This test's purpose is to ensure that when recovering from a suspended state that the PCIe Ethernet device can still connect to the internet. This test puts the device into sleep

mode, wakes itself, and then tries to ping NVIDIA's website. If it can successfully ping NVIDIA's homepage after coming out of a sleep state, the test passes.

5.3.4 Verify PCIe wifi ping

This test is used to test internet connectivity with a PCIe wireless card. This test is very similar to our PCIe Ethernet card test except it pings using the connected PCIe wireless card. If it can ping successfully, then the test passes.

5.3.5 Verify PCIe wifi functions after suspend/resume

This test is similar to our test that ensures that PCIe Ethernet functions properly out of a sleep state. We accomplish this test the same way, we bring the device in and out of sleep mode and ping NVIDIA's website using the wireless PCIe card. If it can successfully ping NVIDIA's website after coming out of a sleep state, the test passes.

5.4 SD Card Reader

The Jetson TX1 supports the use of an SD card reader. To verify that this feature functions properly we run a series of tests with SD cards of varying capacities and different speed classes. These tests are required to be interactive because the user must manually insert and remove their SD cards during the testing process. For these tests the user is prompted to input the speed class of the SD card they want to test, the test then guides the user through a suite of SD card tests that verify SD hot-plug detection, read and write of data, SD card insertion wakes the TX1 from sleep mode, and file transfer rejection while in write protection mode.

5.4.1 Verify SD card detection

Before we begin testing, we need the user to specify the speed of their SD card. The test script does this by prompting the user to insert their SD card, and select its speed class from a list of options. Once selected, the information will be stored in the program and used in all subsequent test cases.

The first test case ensures the card can be detected by the card reader. To do this, the test script checks if there is an SD card directory located in the Linux device directory when the card is inserted. The test will fail if no directory can be found.

5.4.2 Verify SD card read and write

This test is designed to ensure that files can be written to and read from the SD card correctly when the SD card's write protection mode is not turned on. First, the test script prints out a message to remind the user that the SD card should *not* be set in write protection mode. Once the user confirms that their device is not in write protection mode, the script will mount the SD card and copy a file to it. The same file is copied back and renamed, so that it can be compared to the original file. If the file that was copied back is identical to the original, the read and write process completed successfully, and the test passes.

5.4.3 Verify SD card insertion can wake up system

This test case ensures that the system can wake up from low power mode (LP0) mode after inserting an SD card into the SD card reader. First, the script will prompt the user to remove their SD card. After they remove it, the user is informed that the system will temporarily go to sleep after 10 seconds. The user is instructed to wake up the system by inserting their SD card again. After the computer comes out of sleep, the user is asked whether the computer was woken up by the SD card being inserted.

5.4.4 Verify SD card write rejection in write protection mode

The last test is designed to make sure that the system cannot write to the SD card when the SD card is set to write protection mode. To do this, the interactive test script first prints out a message instructing the user to put their SD card into write protection mode. This can be done by enabling a switch located on the physical SD card. Once the user confirms that their SD card is in write protection mode, the test mounts the SD card and tries to copy a file to it. If the copy fails, the test passes indicating write protection is indeed in effect, otherwise, the test fails.

6 Results

The end result of our project is a suite of tests the run in a custom NVIDIA testing framework for NVIDIA to provide to their customers as well as use for their own internal testing. Prior to our project, engineers working at NVIDIA had to manually run these tests tediously step-by-step to ensure that the Jetson T210's hardware was functioning properly. NVIDIA's customers had no reliable testing solution at all. Accounting for the time it could take to gather all of the information about each test, the process of testing the Jetson T210 module could take hours. Now, using the tests that we developed over the course of our time working with NVIDIA, the entire testing process is contained in one simple, user-friendly script that can be run in its entirety in less than 12 minutes. This will without a doubt save NVIDIA and its customers a great deal of time during their bring-up testing process.

7 Conclusion

To summarize, this MQP project focused on analyzing the current manual testing process of the Jetson TX1 and implementing an automated test framework and test modules for various hardware components of the Jetson TX1. We accomplish this by designing and implementing Bash shell test scripts to ensure proper functionality of the Jetson TX1 CPU, GPU, EMC, thermal management, audio, video encoder/decoder, fan pulse-width modulation, LP0, eMMC, Bluetooth, USB, Ethernet, HDMI, and SD card sub-modules. Our tests minimized the manual effort involved in verifying different test cases, and utilized an interactive approach in scenarios where user input is necessary. This report documents our implementation in detail, and proposes future directions to extend, improve, and polish our work.

8 Future Work

The primary goal of this project was to automate the test process of different hardware components of the Jetson TX1 to improve testing efficiency for NVIDIA's internal teams and external partners. As reported in this document, this goal was largely fulfilled by the end of our work contract. Nonetheless, there is still room to extend, improve, and polish our work given more time. Firstly, due to technical challenges and logistic reasons, we did not complete some test cases for some hardware components. For example, one test case of the eMMC asks for verification of file read and write from external sources. One of the specifications for our project is that our test suite must all be run on the TX1 itself. This test, however, requires a host machine to be hooked up to the Tegra X1, so that an external file could be copied. We could have written a test implement to this is through interactive testing, by providing step-by-step instructions to guide the user through the manual process of connecting the TX1 to a PC via USB or through SSH over a network. However, this does not reduce the user's workflow significantly, so it was skipped in the current test implementation. Similarly, for simplicity, the test of read/write of SD card and USB flash drives use small text files instead of Gigabyte-sized files. Ideally, we would run our tests with larger files to ensure complete read/write functionality, although in the interest of time constraints, we only test with small files.

Secondly, some hardware components are not yet covered in our test suite due to various limitations. For example, the I2C, SDIO, and Expansion IO tests were not implemented because they require additional hardware that was not immediately available to us. In the future, the test of these components can be automated by connecting additional hardware following similar methodology used by our completed test scripts.

Lastly, we were limited by time. Some sub-modules proved more difficult to test and required additional information that we were not able to acquire in before the end of our project. An example of this is the Power management integrated circuit sub-module. This module was particularly difficult for us to get information about how to test it. We were unable to gather the information necessary to test it before the end of our project, so this module remains untested by our test suite for now.

References

[1] NVIDIA. NVIDIA: The Visual Computing Company. url: http://www.nvidia.com/content/company-info/introduction-to-nvidia.pdf.

[2] NVIDIA. NVIDIA History. url: http://www.nvidia.com/page/corporate_timeline.html

[3] NVIDIA. NVIDIA Press Release. url: http://www.nvidia.com/object/io_1250488651953.html

[4] NVIDIA. NVIDIA Products. url: http://www.nvidia.com/object/nvs-graphics-cards.html

[5] NVIDIA. NVIDIA News. url: http://nvidianews.nvidia.com/news/nvidia-unveils-tegra-k1-a-192-core-super-chip-that-brings-dna-of-world-s-fastest-gpu-to-mobile

[6] NVIDIA. NVIDIA News. url: http://nvidianews.nvidia.com/news/nvidia-launches-tegra-x1-mobile-super-chip

[7] NVIDIA. NVIDIA GPU. url: http://www.nvidia.com/object/gpu.html

[8] Glmark2. url: https://launchpad.net/glmark2

[9] OpenGL. glxgears url: https://www.opengl.org/

[10] HDMI. What is HDMI technology?. url: http://www.hdmi.org/learningcenter/faq.aspx#1