

NVIDIA: A PINMUX CONFIGURATION

TOOL FOR TEGRA

Jordan Feeley, Yiren Wang, Yuan (Will) Wen

March 3, 2017

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:

Jordan Feeley

Yiren Wang

Yuan Wen

Date: March 2017

Approved:

Professor Mark Claypool, Advisor

This report represents the work of one or more WPI undergraduate students. Submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its web site without editorial or peer review.

Table of Contents

Table of Contents	1
Abstract	4
Acknowledgements	5
1 Introduction	5
1.1 Problem Statement	6
1.2 Process	7
1.3 Results	7
1.4 Roadmap	8
2 Background	9
2.1 NVIDIA's Tegra & Jetson Products	9
2.2 Jetpack for Linux For Tegra	12
2.3 Pinmuxing	14
2.4 Board Configuration File (CFG) Generation Process	15
3 Requirements	17
3.1 Pinmux Configuration	17
3.2 Pad Voltages Configuration	17
3.3 Board Connections and Pin Constraints	18
3.4 Seamlessly Generate CFG Files	18
3.5 Save User's Work	19
3.6 Cross Platform Tool	19
3.7 Package Installation	19
4 Approach	19
4.1 GUI Tools From Other Sources	20
A. Texas Instruments	20
B. Toradex	22
C. GUI Tools From Other Sources Summary	24
4.2 Language Comparison	25
A. Ability to Run Python Scripts	25
B. Cross-platform Capability	26
C. GUI Frameworks	27
E. Language Summary	28

4.3 Understand How CFG Files work	29
A. Understand devmem2	29
B. Find the address location of a register	29
C. Verification of TRM Findings with the default CFG file	31
D. Check the value of the location on the board	31
E. Generate a new CFG file with a slight modification	32
F. Flash the new CFG file to board	32
4.4 Process	34
A. Planned Schedule	34
B. Tools	36
4.5 Design	38
A. GUI Mockup	38
B. Logic Diagram	39
4.6 Coding	41
A. Development	41
B. Third-Party Frameworks and Modules	45
5 Results	49
5.1 Developed Features	49
5.2 Evaluation	53
5.3 Summary	54
6 Conclusion	55
7 Future Work	57
8 References	59
9 Appendices	61
Appendix A Readme.txt	61
Appendix B Third Party Documentation	66
Appendix C Box Diagram	68
Appendix D Mockups	69
Appendix E Validation Rules	71
Appendix F Study of Terminologies	73
Appendix G Tegra default pinmux CFG File	74
Appendix H Screenshots of GUI Tool	76
Appendix I Instructions to generate JSON Data file from Excel Spreadsheet	79

Abstract

NVIDIA produces system on chips called Tegra chips. NVIDIA also produces Jetson development boards, powered by Tegra chips, which are small computer boards that offer pin configurations (known as pinmux) for when users want a pin to provide a different functionality. Currently, pinmuxing is done through a Microsoft Excel spreadsheet. The spreadsheet defines which pins can be configured to what functionality. After configuring, the spreadsheet exports the data to intermediate DTS files. Users must then go through a Python script to convert those DTS files into a board compatible CFG file. Our project is to develop a cross-platform GUI tool with the same functionality as the spreadsheet, plus new features such as pinmux constraints depending on user attached boards. After gathering the requirements, we created a GUI mockup and logic flow diagram. The GUI tool was primarily written in JavaScript and involved many weeks of iterations with our peers. At the conclusion of our project, we were able to deliver a functional web app that gives users the ability to configure pins, drag and drop connectors, and seamlessly export CFG files.

Acknowledgements

We would like to thank the following for their help and guidance during this project.

Stephen Warren - Requirements, feedback, clarifications

Christian Gonzalez - L4T setup, building, flashing

Mark Claypool - Software development advice

Richard Shen - Excel spreadsheet and hardware questions

Winnie Hsu - Logistics and networking

Chris Freeman - Git processes and commands

Sumathi Natarajan - Internal JavaScript code reviewer

Lane Harrison - External web development suggestions

1 Introduction

NVIDIA is one of the tech world's leading companies in graphics processing hardware, artificial intelligence (AI), and autonomous vehicles. Currently NVIDIA's Tegra SoCs (System on a Chip) provide additional mobile capabilities with power and efficiency [Introducing NVIDIA]. The Tegra TX1 SoC is one of NVIDIA's advanced mobile processors. NVIDIA also produces Jetson development boards, powered by Tegra SoCs. These boards are aimed at developers, where they can express creativity and create projects such as autonomous navigation to deep-learning analytics.

The Tegra SoC has over 400-pins that connect to the various boards. Users can configure the Tegra pins to their own desired usage. In order to flash a board with customized pin configurations, the developer must use a Microsoft Excel spreadsheet to make changes, then go through a series of steps to flash the board, discussed in section 2.5.

1.1 Problem Statement

Our project is intended to provide a new cross-platform pinmux Graphical User Interface (GUI) tool that has all the same functionality as the Excel spreadsheet along with an easy to use interface and additional configuration options to help NVIDIA personnel and consumers configure their Tegra SoCs. The pinmux tool provides a GUI for configuring pinmux settings, validating user options, and specifying pad group voltage. An existing Tegra pinmux Python script is also used to export a CFG file.

1.2 Process

The development process started with mockups, transitioned into coding and redesigning based on feedback, and ended with testing and packaging. Before starting the mockups, we familiarized ourselves on tools such as Trello and Gerrit that would help organize our development process. We then drafted a GUI mockup along with a logic flow diagram. After review from NVIDIA personnel, we started coding. We implemented groups of features and presented and edited them based on feedback. Near the end of our coding cycle, we tested our the CFG files generated from our tool and the Excel spreadsheet by comparing a text difference. Lastly, we packaged our web app as distributables for Mac, Linux, and Windows Operating Systems through the use of a software package called Electron. Development comprised of initial mockups and diagrams, developing the requirements through weekly iterations, and redesigning based on continuous feedback.

1.3 Results

The Tegra Pinmux GUI Tool we developed supported all pre existing functionality as well as added new features that will help the overall user experience. Features that we were able to develop include full functionality of the existing pinmux spreadsheet, constraints based on a boards connected in a device tree, import/export of current work, and pad group filtering. The full functionality of the existing spreadsheet includes modifying the same pinmux settings as before, as well as exporting to a DTS file. Constraints is a new feature that prevents specified modifications of pin settings based on what boards were connected to the Tegra SoC. Import and

export of current work is also a new feature that allows users to share their work as well as save work for later edits. Pad group filtering minimizes a large table and only show the pad groups the user selected.

1.4 Roadmap

In the following sections, we discuss necessary background information, requirements of our GUI tool, the approach we took to design a tool that meets those requirements, and the end results we were able to develop. Background (Chapter 2) provides information about the Tegra SoC, Jetson development boards, pinmuxing, and NVIDIA's current process of pinmuxing on a Jetson board. Requirements (Chapter 3) lists our initial backlog of features that the GUI tool must have based on NVIDIA requirements. Approach (Chapter 4) documents the steps we took in order to meet those requirements. Results (Chapter 5) talks in detail, along with screenshots, about each feature we integrated in our GUI tool. Conclusion (Chapter 6) summarizes the whole project motivation, statement, and the results we accomplished.

2 Background

Background information about NVIDIA's Tegra and Jetson products is provided in this chapter. After the products are introduced, we show the initial flashing of Jetpack for L4T onto a new TK1 Development Kit. Pinmuxing is also defined as well as its purpose in relation to the Jetson Development Kit. The process of flashing a pinmux setting, through a Microsoft Excel spreadsheet, onto a development board is also shown.

2.1 NVIDIA's Tegra & Jetson Products

NVIDIA is an American technology company based in Santa Clara, California. NVIDIA designs graphics processing units (GPUs) for the gaming market, as well as system on a chip units (SOCs) for the mobile computing and automotive market [Introducing The Tegra]. More recently, NVIDIA has moved into the mobile computing market, where it produces Tegra mobile processors for smartphones and tablets. For example, the Google Pixel and NVIDIA Shield Tablet are both powered by NVIDIA's Tegra SoCs. Tegra mobile processors include the TX1 and TK1 models. NVIDIA also produces Development Kits that are development boards powered by a Tegra SoC. An example is the TX1 Development Kit and its components is as follows:

Tegra TX1 SoC: The TX1 SoC (Figure 2.1.1) is a System on a Chip (SoC) developed by NVIDIA for mobile devices such as smartphones. This SoC contains 256 GPU Cores and a Quad-core 64-bit ARM A57 CPU.

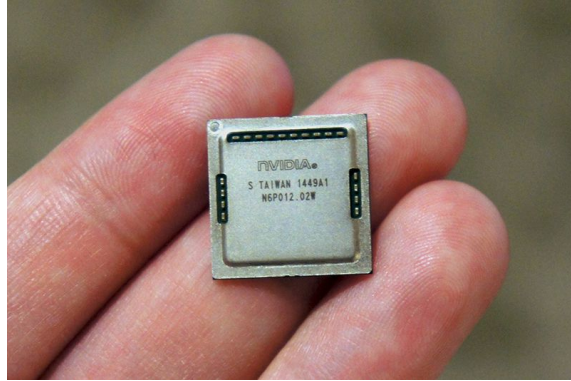


Figure 2.1.1 A TX1 Tegra chip

Module: The TX1 module (Figure 2.1.2) is for the TX1 SoC, which is located in the center of the module. The module includes power management controllers, external memory (4GB LPDDR4), storage (16 GB eMMC flash), and WiFi/Bluetooth controllers.

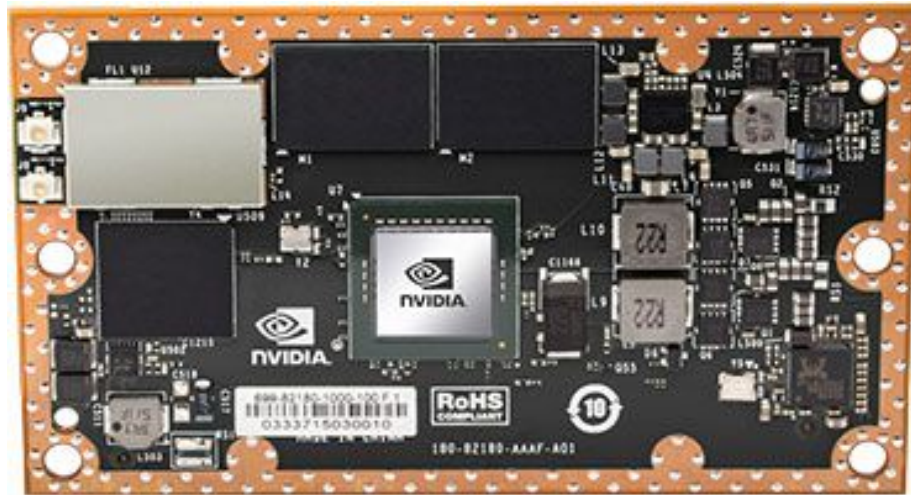


Figure 2.1.2 Jetson TX1 Module

TX1 Jetson Baseboard: The Jetson baseboard can be seen in Figure 2.1.3. Users can use their own board, as long as it connects to the Jetson TX1 module correctly. The baseboard contains peripheral connectors such as general purpose input output (GPIO) expansion headers, WiFi antenna, 19v AC power jack, and camera headers.

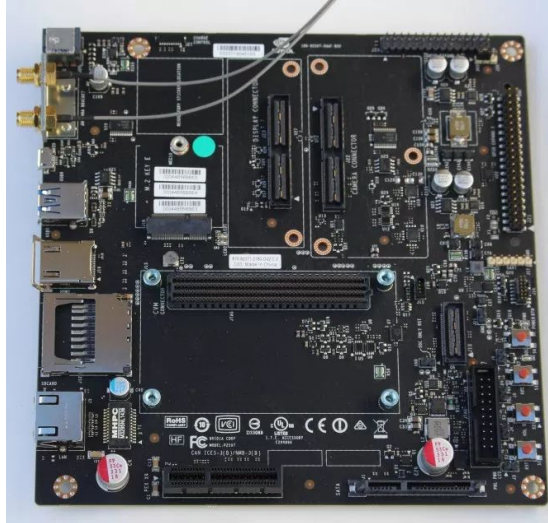


Figure 2.1.3 TX1 Jetson Baseboard

TX1 Jetson Development Kit: The Jetson Development Kit can be seen in Figure 2.1.4. The Jetson Development Kit combines the Tegra SoC, Jetson module, and Jetson board. This kit is sold to users and developers to use for their own applications.

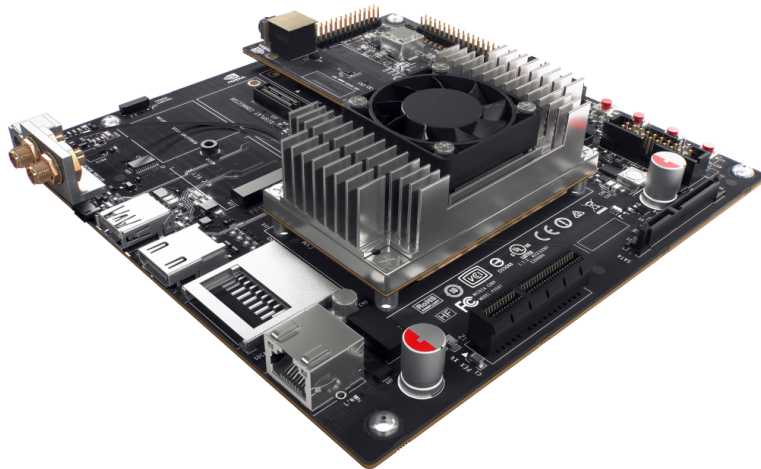


Figure 2.1.4 TX1 Jetson Development Kit

2.2 Jetpack for Linux For Tegra

The Linux For Tegra (L4T) package is Ubuntu OS with NVIDIA drivers, packages, and programs specifically designed for Jetson Development Kits (TK1, TX1 boards). Jetpack for L4T includes all the L4T board support packages plus additional software such as CUDA, TensorRT, and cuDNN. As defined on the NVIDIA website, “the Jetson Development Pack (Jetpack) for L4T is an on-demand, all-in-one package that bundles and installs all software tools required to develop for the NVIDIA® Jetson Embedded Platform.”[JetPack for L4T] Users who have a host computer on Ubuntu 14.04, can flash Jetpack for L4T on their target development board. In Figure 2.2.1, a host machine running Ubuntu 14.04 is flashing Jetpack for L4T onto a new TK1 development board. After the flash, the target TK1 can then be independently booted and runs L4T OS. Figure 2.2.2 shows the target device, in this case a TK1, running some sample applications from the Jetpack package. Jetpack for L4T includes the board support package, additional software, and samples to help developers quickly start up with their Jetson Development Kits.

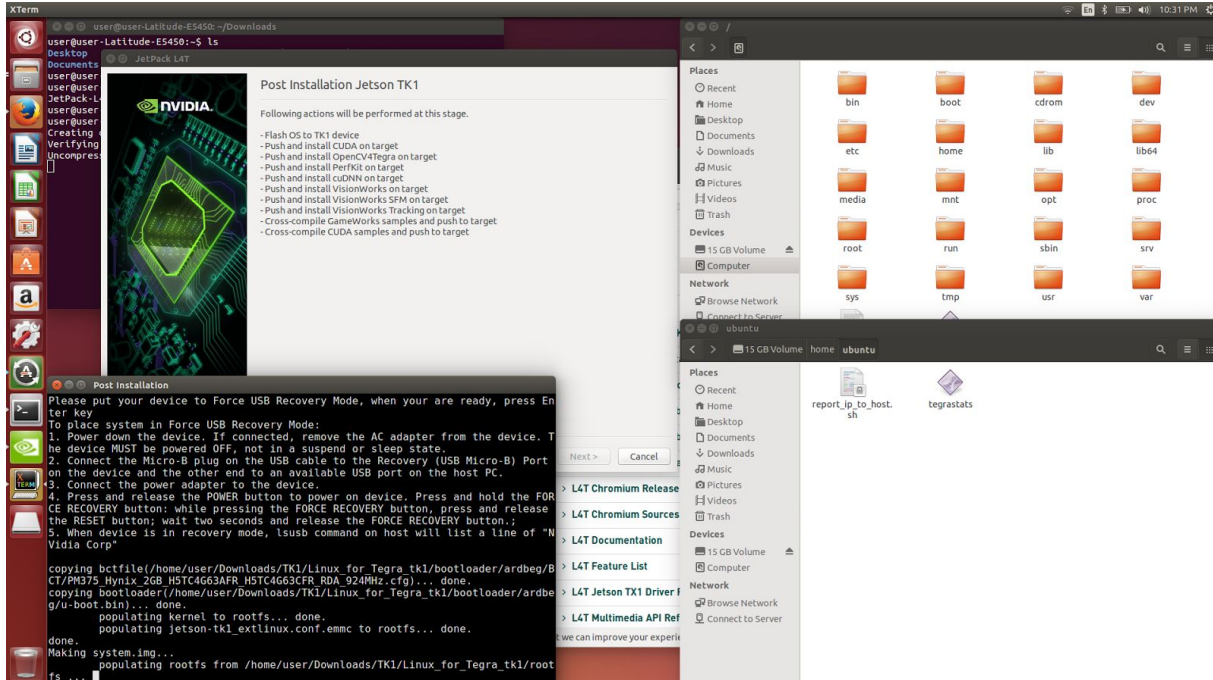


Figure 2.2.1 Host Machine (Ubuntu 14.04) flashing Jetpack L4T onto a TK1.

A post installation screen is present, summarizing all the packages flashed on the TK1.

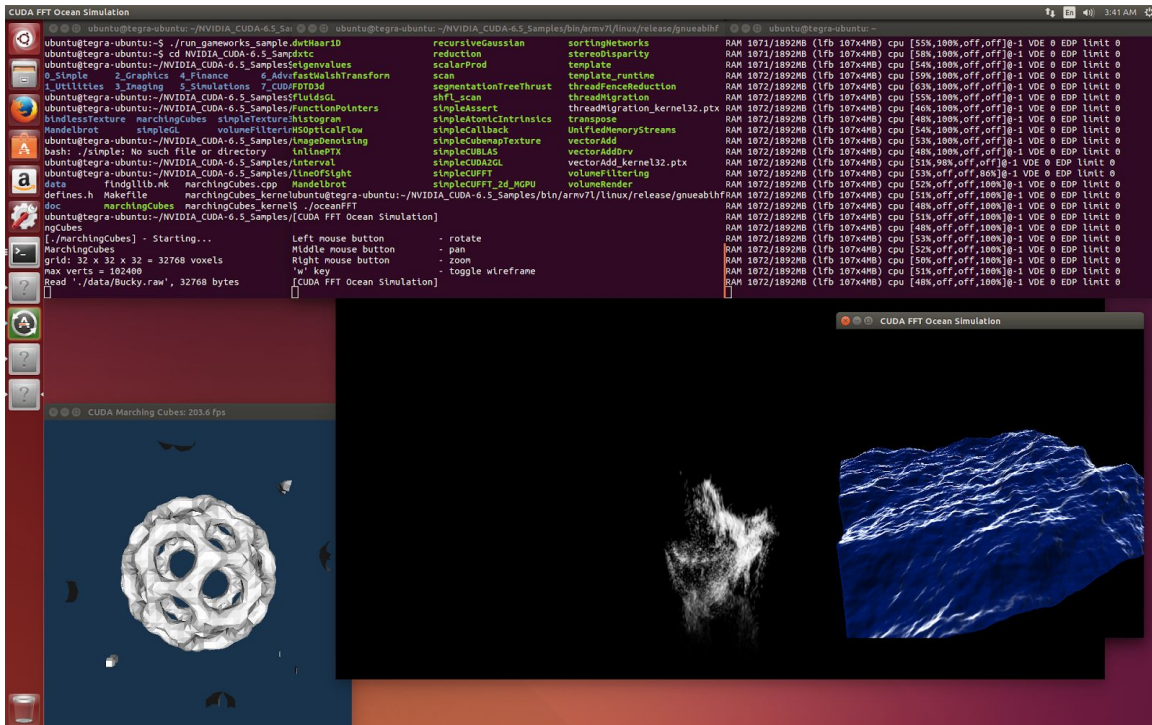


Figure 2.2.2 TK1 running various Jetpack application samples including a CUDA Simulation

2.3 Pinmuxing

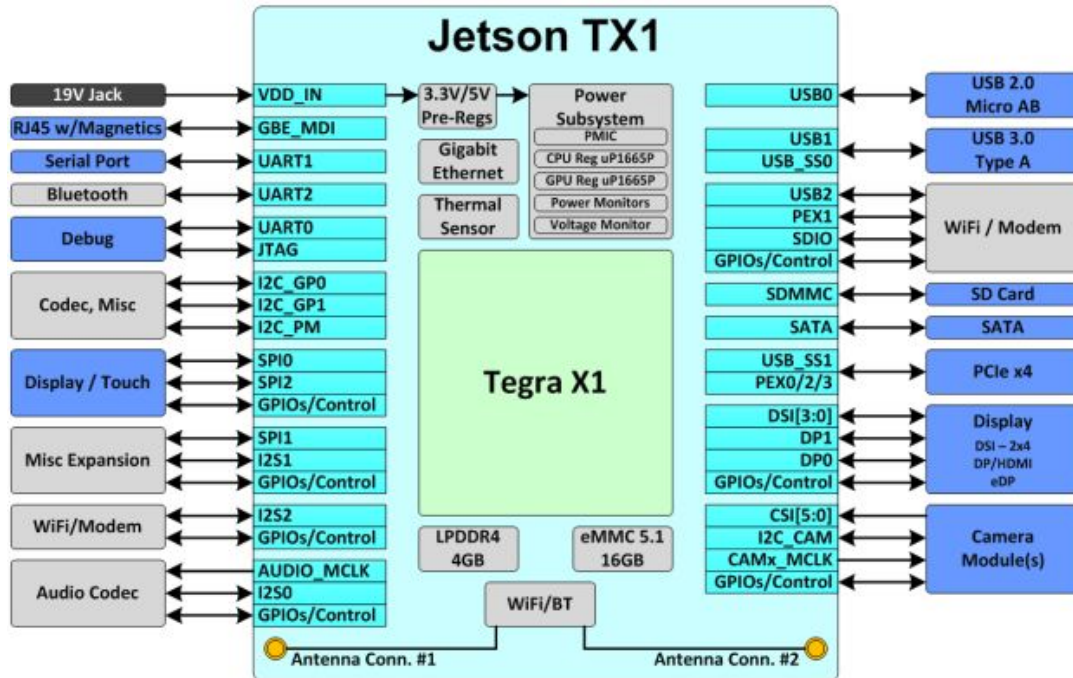


Figure 2.3.1 Jetson TX1 block diagram

Pinmux is the software controlled configuration of a pin's functionality. To help understand the concept in more detail, Figure 2.3.1 shows a block diagram of a Jetson TX1 Development Kit. Blocks on the perimeter indicate components on the Jetson Carrier Board (CVB). For example, the 19v AC power jack, WiFi antenna, and SD card slot are apart of the Jetson board. The big teal block in the center is the Jetson Carrier Module (CVM), which houses the Tegra TX1 SoC (in light green). The CVB has peripheral headers (groups of pins) such as USB, HDMI, DSI, GPIO that connect to the CVM. The connections are made through physical pins touching on the CVM and CVB. Inside the CVM, there are also pins that must communicate with the TX1 SOC.

Pins can generally be left to serve their default usage, serve as a GPIO, or be unused. If we take a look at the SATA pin group on the middle right hand side, all the pins in that group can be pinmuxed to be a default SATA usage, set as a GPIO pin, or set to unused. A developer would select pinmux to GPIO if they do not plan to use the SATA connection and instead wants to use the pin for GPIO. If a developer does not plan to use that pin at all, then the SATA pin can be pinmuxed to be unused.

Along with pinmux configurations, developers can also control the upper bound of voltage outputted from a pin group. This is called controlling the pin group's pad voltage. For example, by setting the pad voltage in the pin group "UART1" from 1.5v to 3.3v, the output high voltage of all pins in that pin group are changed from 1.5v to 3.3v.

2.4 Board Configuration File (CFG) Generation Process

Previous to our project, the users generated CFG files using an Excel spreadsheet. In order to flash a board with the customized pin configurations, the user used the spreadsheet to make changes on pinmux settings, then executed Visual Basic macros to generate intermediate device tree source (DTS) files. The DTS files were moved to a Linux platform and the user executed the dos2unix command to change the line endings to Unix format. Finally, the user executed a Python script, inputting in the DTS files, to generate configuration (CFG) files. The new CFG files could then be moved to a directory on the host machine and be flashed onto the board. A picture of this process is in Figure 2.4.1.

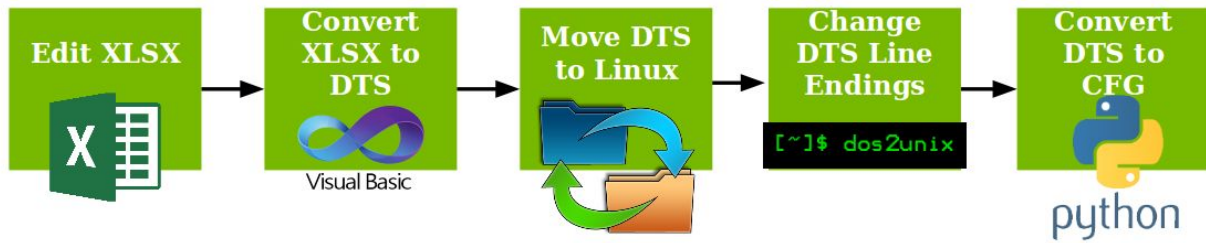


Figure 2.4.1 CFG File Generation Process Graph

This long process not only requires lots of user interaction, such as moving the files to the appropriate OS, but also creates the intermediate DTS files that have no use to the end user. Since the Excel spreadsheet can only be opened on a Windows computer, while the CFG files must be flashed on a Linux host machine, the process of transferring files between operating systems results in extra work for a user.

3 Requirements

The Requirements section states what features were required for the Tegra Pinmux GUI Tool, as specified by NVIDIA developers, as well as the reasons for each of them.

3.1 Pinmux Configuration

NVIDIA used an Excel spreadsheet to configure pins and generate CFG files that could be flashed onto the Tegra board. Our tool should contain all the same functionalities that the Excel spreadsheet had. The spreadsheet contained hundreds of rows, each row representing a pin and the pinmux options it had. The user selected the pin functionalities in the dropdown menus. The spreadsheet also had a data validation function, which would mark the cells as red if the pin was configured incorrectly. Our tool should configure the same amount of pins with the same options as the spreadsheet, and should also detect and show errors if the pins were assigned wrong configurations.

3.2 Pad Voltages Configuration

Another requirement was to include pad voltages in our tool. The spreadsheet had voltage configuration along with the pinmux settings. Pad voltages are only configurable for the pad groups (groups of pins with the same default functionality.) A pad group should only be assigned one voltage, and this voltage will affect all the pins in that group. Therefore our tool should have a separate tab for pad voltages, and changing the voltage would affect all the pins under that pad group in the pinmux settings.

3.3 Board Connections and Pin Constraints

A functionality that the Excel spreadsheet did not have, was to constrain pinmux options based on board connections. While a Tegra SoC is the initial connection with no constraints applied, when a user adds a module and a board, the GUI tool should constrain some pins to a specific usage. The specific constraints applied would depend on data files read in by our tool. The tool needs to include a tree representation of which boards are connected, and constrain pins depending on those board connections.

3.4 Seamlessly Generate CFG Files

The new GUI tool should generate the same CFG files as the spreadsheet does to avoid flashing incorrect pinmux settings to the board. In Section 4.3, we prove that if two CFG files are similar, then they result in the same pinmux settings. Along with the requirement to produce the same CFG, the GUI tool must also remove intermediate DTS files. Abstracting DTS from users will reduce much of the overhead (refer back to Figure 2.4.1) to generate CFGs. The new CFG file generation method is shown in Figure 3.4.1.

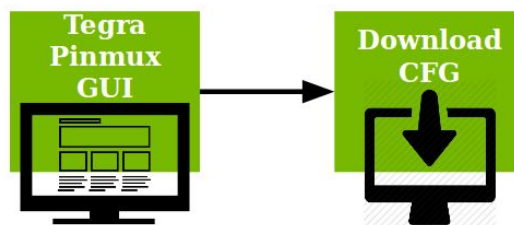


Figure 3.4.1 New CFG File Generation Process Graph

3.5 Save User's Work

A feature in any tool would be to the ability to save and resume one's work. In relation to the Tegra Pinmux GUI Tool, saving work would save all the pinmux changes the user has made. Importing from a save file bring the exact same state as before.

3.6 Cross Platform Tool

An old issue about the spreadsheet was that it was in .xlsx format and had only worked on Windows. Our new tool should be cross-platform to avoid Windows dependency. It should be able to run on Windows, Linux and Mac OS platforms and perform the same functionalities.

3.7 Package Installation

Finally, we need to combine our tool into a package for easy user installation. If we decided to distribute a raw web application with a locally hosted server, then the user needed to install the Node.js and Node module dependencies on his own machine. The web tool needs to be packaged into a simple installer for the user to avoid complicated dependency installation instructions.

4 Approach

Approach documents the steps our team took in order to meet our requirements. We first analyzed GUI tools from other similar embedded system manufacturers. Then, given many different languages to write a GUI tool in, we compared and contrasted the languages. A very

simple pinmux flash was done on a Jetson board to verify a pin’s change in configuration. We then familiarized ourselves with Trello, our scrum board, and Gerrit, our code review software. The last step before starting to code, our team created a GUI mockup and logic flow diagram.

4.1 GUI Tools From Other Sources

Before starting the Tegra Pinmux GUI tool design, we analyzed GUI tools from two other similar products during our research. The main one was from Texas Instruments, and the other one was developed by Toradex.

A. Texas Instruments

Texas Instruments (TI) has two versions of their pinmux GUI tool - one for desktop and one on the web. As we can see in Figure 4.1.1, the web version is more up to date and supports the OS X platform while the desktop version is archived and only supports Windows and Linux. Seeing the cloud version as more up to date, we inspected TI’s GUI tool and found various design patterns that we may want to consider.

Part Number	Buy from Texas Instruments or Third Party	Status	Current Version	Version Date	Host	OS	Description
PINMUXTOOL_DESKTOP_PREVIOUS: Archived version of desktop Pin Mux tool supporting AM35x, AM/DM37x, DM816x. V4 is recommended for new Sitara designs	Free View	ACTIVE	v2 and v3	previous		Windows, Linux	Standalone desktop versions of the tool. Device and OS support vary by version
PINMUXTOOL-V4-CLOUD: Pin Mux tool for AM335x, AM437x, AM572x, AM571x, CC3200, F2807x, F2837xD, F2837xS, MSP432, TM4C123x, and TM4C129x	Free View	ACTIVE	v4	09 Sept 2015	Windows, Linux, OS X		Browser-based tool access via TI Cloud Tools portal. Automatic solving, high-level requirements entry for configuring device mux settings

Figure 4.1.1 Two versions of TI’s pinmux GUI tool
First one is an old version to run on Windows and Linux,
while the second tool runs on the cloud and is version 4.

In Figure 4.1.2, we can see that TI's Cloud based GUI tool first starts out with allowing the user to create a new pinmux design by selecting from a variety of devices, parts and packages, or to load a previous existing design.

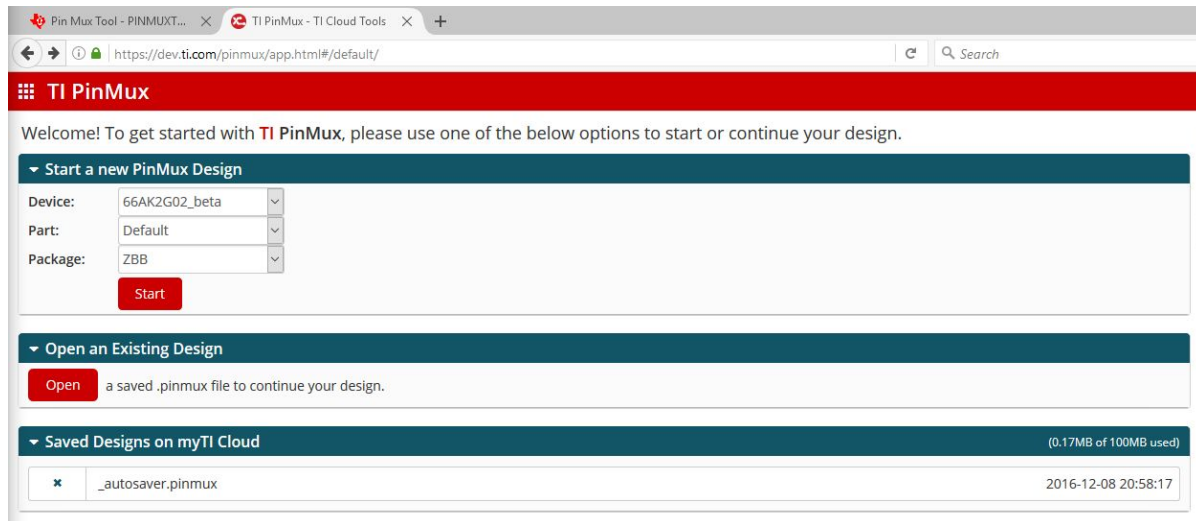


Figure 4.1.2 TI pinmux selection page

Start screen of TI's GUI Tool. Able to select from a variety of devices, parts, and packages or load a pre-existing design.

Moving on to the actual pin configuration screen, in Figure 6.1.3, the layout was split into three sections: Peripherals, Requirements, and Output. The Peripherals section has the domain of all possible pins to configure on the TI Board; Requirements is the section to set the pin constraints, and Output is to view the design summary, generate files (.c and .h files to simulate your requirements), and visually see the pin layout.

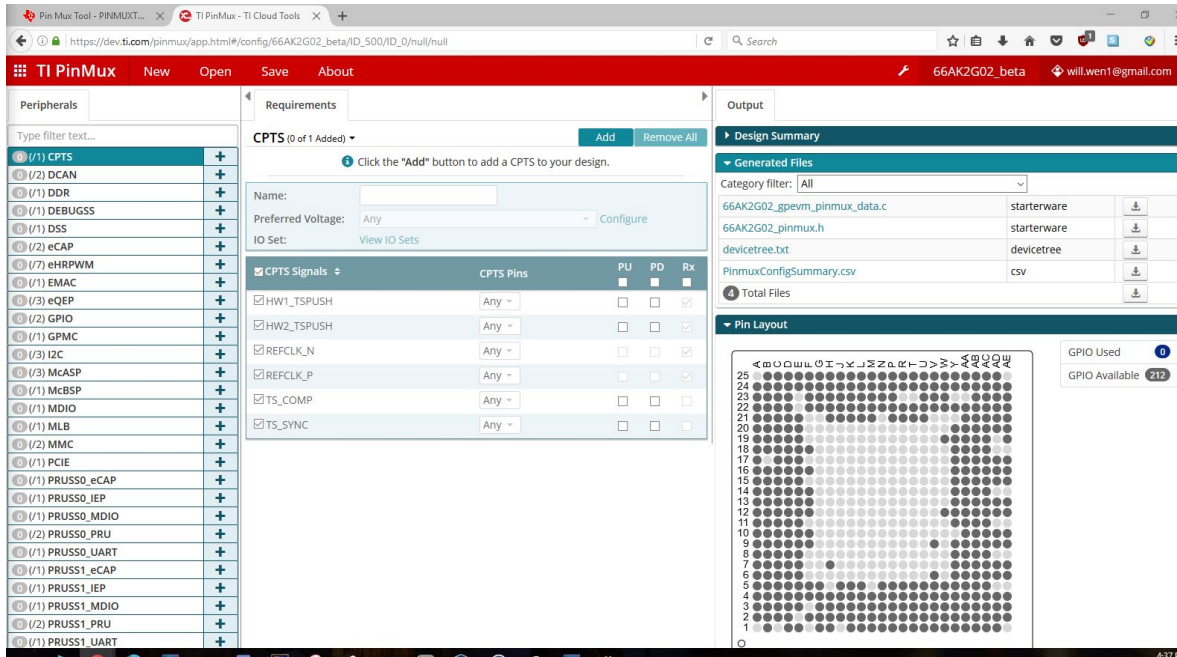


Figure 4.1.3 TI pinmux configuration page

Main configuration page contains peripherals (contains pins that can be modified) on the far left, pin settings in the middle, and a design summary with relevant device files and a pin layout.

Viewing TI’s GUI tool from the front end gave us information about important sections to see from an engineer’s standpoint. It also showed that TI has pushed forward to use a web-based platform for more OS compatibility. These observations influenced our decision to make a web app vs a desktop app, and design our GUI mockup,

B. Toradex

Another GUI tool was made by Toradex, a company that produces System on Modules (SoM). On the download page of their website (Figure 4.1.4), Toradex only has a Windows version of their pinout designer, being as current as December 3rd, 2016. Although Toradex did

not have a cloud platform, we decided to investigate into their GUI tool and see for layout similarities with TI.

#	File Name	Description	Revision	Version	File Size	Submitted Time	Current
1	toradex_pinout_designer_v1.3.4.0_2016-11-03.zip	Version 1.3.4.0	12		7.62 MB	2016-11-03 19:41	Yes
2	toradex_pinout_designer_v1.3.3.0_2016-09-30.zip	Version 1.3.3.0	11		7.61 MB	2016-09-30 20:45	-
3	toradex_pinout_designer_v1.3.2.0_2016-03-31.zip	Version 1.3.2.0	10		7.62 MB	2016-03-31 14:33	-
4	toradex_pinout_designer_v1.3.1.0_2016-02-10.zip	Version 1.3.1.0	9		7.62 MB	2016-02-10 21:46	-
5	toradex_pinout_designer_v1.3.0.0_2015_07_16.zip	Version 1.3.0.0	8		7 MB	2015-07-16 14:56	-
6	toradex_pinout_designer_v1.2.0.0_2015_05_13.zip	Version 1.2.0.0	7		8.42 MB	2015-05-13 21:03	-
7	toradex_pinout_designer_v1.1.0.0_2015_03_30.zip	Version 1.1.0.0	6		8.41 MB	2015-03-30 20:46	-
8	toradex_pinout_designer_v1.0.0.0_2015_02_10.zip	Version 1.0.0.0	5		6.9 MB	2015-02-10 23:47	-

Figure 4.1.4 Toradex Pinout Designer versions

Only containing one designer for windows, but went through as many as 12 revisions.

Very up to date with the last version being uploaded at 11/03/16.

The Toradex Pinout Designer shares all the three similar categories - Peripherals, Requirements, and Output, though named differently and on different degrees of detail. The Peripherals category nearly mirrors the TI's, but the Requirements on the Toradex does not allow easy on-the-fly editing of the pin values. The Output category also differs because while Toradex shows the pins being used and pins in conflict, it does not show the pin layout nor the design summary while TI does.

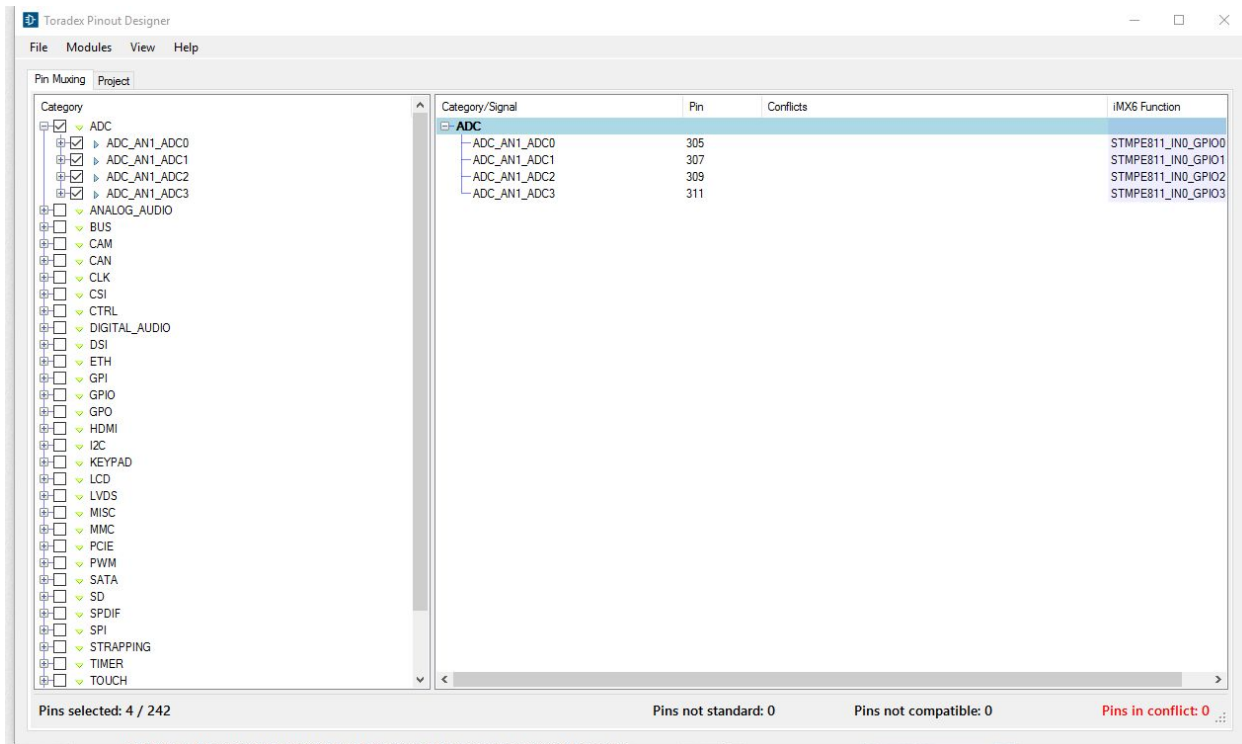


Figure 4.1.5 Toradex Pinout Designer Program

Latest 1.3.4.0 version downloaded from the Toradex website. Left side is similar to TI’s peripherals panel, and the center is also similar to TI’s pin setting panel. The difference is the scope of the design summary panel. Toradex has the bottom row for a summary, while TI has a dedicated column with a Pin Layout diagram.

C. GUI Tools From Other Sources Summary

After looking at the pinmux GUI tools from TI and Toradex, we concluded that a web-based tool is more scalable on different OSs. We also advised that our tool should be split up into three categories: pin selection, pin editing, and pin summary.

4.2 Language Comparison

There were several programming languages that we could choose from to implement our GUI tool: Python, Java, C++ (with Qt) and JavaScript (with a backend Node.js). We compared them from different aspects in the following sections, and discuss our evaluation below:

A. Ability to Run Python Scripts

NVIDIA has already developed some upstream pinmux tools to configure the pins and boards, and those scripts were written in Python [Tegra-Pinmux-Scripts]. Our GUI tool should be able to execute some of these scripts directly in order to connect with the board. We found out that all four languages can run Python scripts, and the main commands (or libraries) for each of them are listed in the Figure 4.2.1. Most notably, writing our tool in Python means the user already has Python installed. However, if we developed in Java, C++ , or JavaScript, creating a child process needs to assume a user has Python installed, otherwise the OS won't find Python.

Python	<ol style="list-style-type: none"> 1. <code>from sys import path</code> <code>path.append(path_to_file)</code> // the above two lines are omitted if file is in the same directory <code>import [file name]</code> 2. <code>os.system("script.py 1")</code>
Java	<code>Process p = Runtime.getRuntime().exec("python path/script.py");</code>
C++ with Qt	<ol style="list-style-type: none"> 1. Using the library “py_embed” 2. Using the Python/C API
Node.js (JavaScript)	<ol style="list-style-type: none"> 1. Using the library “python-shell” (MIT license) 2. <code>var spawn = require("child_process").spawn;</code> <code>var process = spawn('python', ["path/to/script.py", arg1, arg2, ...]);</code>

Figure 4.2.1 Comparison: how to run python scripts

B. Cross-platform Capability

The GUI tool we would create should be able to run on different operating systems, including Windows, Linux, and Mac OS. Linux, for example, has different distributions with variance in compilers and libraries, so we needed to pay attention to the libraries we wanted to use and test the application on all the platforms in the end. Figure 4.2.3 lists the basic requirements for a platform to run the GUI tool.

Python	Python 2 or 3 installed
Java	JRE support (has different versions)
C++ with Qt	G++ (GNU C++ Compiler) installed
Node.js (JavaScript)	A web browser (Chrome, IE, Firefox, etc.) and Node.js installed

Figure 4.2.3 Comparison: platform requirements

Meanwhile, operating systems have different UI guidelines. A cross-platform application will have similar functions on all the OSs, but the UI design can be very different depending on native UI components of the platform. If we used Python, Java or C++ to develop the GUI tool, we must adjust the UI appearance and even the flow of the application to fit different platforms.

Using JavaScript along with a web browser may not be a common solution for implementing a GUI tool, but it does not have the problem of varying UI designs. Since the UI is always loaded inside a web browser, it can have a common look and the user will not feel different. The only problem left would be to scale the web pages for different sizes of screen, while most components and features can still operate in the same way.

C. GUI Frameworks

We also looked into the GUI frameworks these languages support. Using existing powerful frameworks could make the UI look good as soon as we came up with the general design, and save our time for adjusting the UI components. The frameworks are listed in Figure 4.2.4.

Python	Cross-Browser: PyJamas Cross-Platform: AVC, Kivy, etc.
Java	<ol style="list-style-type: none"> 1. Swing, SwingX 2. Window Builder (Eclipse plug-in) 3. Pivot
C++ with Qt	<ol style="list-style-type: none"> 1. Qt
JavaScript	<ol style="list-style-type: none"> 1. Bootstrap 2. Electron (Framework for desktop application)

Figure 4.2.4 Comparison: GUI frameworks

E. Language Summary

All of the four languages are powerful enough and have the functionality required for our GUI tool. They are able to run the pinmux Python scripts, work on multiple platforms and have strong GUI framework support. Thinking about the simplicity of UI design, we decided to use JavaScript with a web browser because it was the easiest way to make our application cross-platform. Also the code on the client and server side would be written in JavaScript, as Node.js runs JavaScript. However, using Python might be preferred by NVIDIA since their Linux 4 Tegra upstream team had more experience with it. In that case we needed to study Python in order to understand the existing scripts, and learn about GUI design using Python.

4.3 Understand How CFG Files work

Before starting working on our tool, we would like to understand how the CFG files work and get familiar with the board and the flashing process. Our first task was to try change the pinmux configurations ourselves. We could change one pin configuration in the spreadsheet, flash the new CFG file generated to the board, and verify that the pin had been changed. The process of completing this task is documented below.

A. Understand devmem2

“devmem2” is a program pre-installed with the Jetson package, so it can be found on the Ubuntu OS installed in the board. Given an address location in the memory, the program can return a 4 byte value at that location.

B. Find the address location of a register

In order to get the address location to pipe into devmem2, we had to read the Technical Reference Manual (TRM) of the TX2 SoC to find where a specific pin is located in memory. In this task we used GPIO_SW1_PFF1 as an example. We had to check the values in both Figure 6.3.1 and 6.3.2.

AMAP Aperture Name (Description)	Parent Aperture	Address Start	Address End	Locality	List of headers corresponding to this aperture
PADCTL_A12	AON_PADCTL_0	0x0c301000	0x0c301fff		arpadctl_SYS.h, arpadctl_SYS_scr.h

Figure 4.3.1 Documentation showing where in memory the PADCTL_A address starts

14.32.31 PADCTL SYS Registers

14.32.31.1 PADCTL_SYS_GPIO_SW1_0

Offset: 0x0 | Read/Write: R/W | SCR Protection: SCR_GPIO_SW1_0 | Reset: 0x00000458

Bit	Reset	Description
12	DISABLE	E_SCHMT: 0 = DISABLE 1 = ENABLE
11	DISABLE	E_OD: 0 = DISABLE 1 = ENABLE
10	HSIO	GPIO_SF_SEL: 0 = GPIO 1 = HSIO
8	DISABLE	E_LPDR: 0 = DISABLE 1 = ENABLE
6	ENABLE	E_INPUT: 0 = DISABLE 1 = ENABLE
4	TRISTATE	TRISTATE: 0 = PASSTHROUGH 1 = TRISTATE
3:2	PULL_UP	PUPD: 0 = NONE 1 = PULL_DOWN 2 = PULL_UP 3 = RSVD
1:0	RSVD0	PM: 0 = RSVD0 1 = RSVD1 2 = RSVD2 3 = RSVD3

Figure 4.3.2 Documentation showing the specific register offset of the address starting location

Along with getting the register's address location, the TRM allows us to interpret the value bit by bit. For example, bit 4 represents the tri-state value. If bit 4 is a 0, the tri-state is passthrough (disabled); if it is a 1, the tri-state is enabled. This is also the bit we specifically targeted in the spreadsheet.

C. Verification of TRM Findings with the default CFG file

From the TRM, we see that the GPIO_SW1 pin has an offset of 0x0 from the starting address location of PADCTL_SYS, which is 0x0c301000. Adding the two together results in GPIO_SW1 being located at 0x0c301000.

We can now verify the address location by checking the default CFG file that is flashed onto the TX2 Board. To find out which CFG file is used for pinmux configuration, we checked the flashing configuration file “P2771-0000.conf.common”. The line 217 of the file said PINMUX_CONFIG = "tegra186-mb1-bct-pinmux-quill-p3310-1000-a00.cfg", so we would check this CFG file for the pin.

This CFG file can also be found in Appendix G. The line 27 in that CFG file is “pinmux.0x0c301000 = 0x00000058 #gpio_sw1_pff1”. The words after “#” is the comment of the line, so this line means the location 0x0c301000 is set to 0x58. The word “gpio_sw1_pff1” verified our assumption, which told us that 0x0c301000 is the location for the pin GPIO_SW1_PFF1.

D. Check the value of the location on the board

```
ubuntu@tegra-ubuntu:~$ sudo ./devmem2 0x0c301000
/dev/mem opened.
Memory mapped at address 0x7f7ff21000.
Value at address 0x0C301000 (0x7f7ff21000): 0x00000058
```

Figure 4.3.3 Running devmem2 on the TX2 Development board, retrieving the value at address location 0x0c30100.

As shown in the Figure 4.3.3, running devmem2 on a newly flashed board for the given location 0x0c301000 returned the expected result 0x58. Since the board was flashed with the default CFG file we found, the value matched the one in the file.

E. Generate a new CFG file with a slight modification

Next we opened the spreadsheet T186_customer_pinmux.xlsm and switched to the “P3310-1000-A0-A2-CVM-Config” tab (Figure 4.3.4), because the name matched the PINMUX_CONFIG value in the file p2771-000.conf.common.

0_DAT7		vddio_sdmme2_h		CZ	BDSMEM_CFCR90_VXVDP1P1P1VC	20K	pd	EQOS_TXC
0_CMD		vddio_sdmme2_h		CZ	BDSMEM_CFCR90_VXVDP1P1P1VC	20K	pu	EQOS_TD0
0_DAT0		vddio_sdmme2_h		CZ	BDSMEM_CFCR90_VXVDP1P1P1VC	20K	pu	EQOS_TD1
E3301-1000-PB-QSPI-Config		P3310-1000-A0-A2-CVM-Config			P3310-1000-A03-CVM-Config		P3310-1000-C0-C	

Figure 4.3.4 Spreadsheet tabs showing the different models of CVMs

We changed the gpio_sw1_pff1 pin from tri-state enabled to disabled. Then we generated three DTS files using the spreadsheet macros, copied them to Linux, ran dos2unix to change the line endings to Unix format, and then ran the Python script DTS2CFG.py to generate the CFG file.

F. Flash the new CFG file to board

We replaced the old CFG file in the directory with the new edited one. Figure 4.3.5 below is a text difference between the two CFG files.

115. pinmux.0x0c303018 = 0x00000400; # can0_dout_pz2: can0, tristate-disable, input-disable	315. pinmux.0x0c303018 = 0x00000400; # can0_dout_pz2: can0, tristate-disable, input-disable
116. pinmux.0x0c303020 = 0x00000458; # can0_din_pz3: can0, pull-up, tristate-enable, input-enable	316. pinmux.0x0c303020 = 0x00000458; # can0_din_pz3: can0, pull-up, tristate-enable, input-enable
117. pinmux.0x0c303028 = 0x00000452; # can_gpio0_paa0: dmic5, tristate-enable, input-enable	317. pinmux.0x0c303028 = 0x00000452; # can_gpio0_paa0: dmic5, tristate-enable, input-enable
118. pinmux.0x0c303030 = 0x00000402; # can_gpio1_paa1: dmic5, tristate-disable, input-disable	318. pinmux.0x0c303030 = 0x00000402; # can_gpio1_paa1: dmic5, tristate-disable, input-disable
119. pinmux.0x02440020 = 0x00000550; # dp_aux_ch0_hpd_pp0: dp, tristate-enable, input-enable, io_high_voltage-disable	319. pinmux.0x02440020 = 0x00000550; # dp_aux_ch0_hpd_pp0: dp, tristate-enable, input-enable, io_high_voltage-disable
120. pinmux.0x02440030 = 0x00000560; # hdmi_cec_pp2: hdmi, tristate-disable, input-enable, io_high_voltage-enable	320. pinmux.0x02440030 = 0x00000560; # hdmi_cec_pp2: hdmi, tristate-disable, input-enable, io_high_voltage-enable
121. pinmux.0x02434058 = 0x00000000; # gpio_wan1_pb4: rsvd0, tristate-disable, input-disable	321. pinmux.0x02434058 = 0x00000000; # gpio_wan1_pb4: rsvd0, tristate-disable, input-disable
122. pinmux.0x02434050 = 0x00000058; # gpio_wan2_pb5: rsvd0, pull-up, tristate-enable, input-enable	322. pinmux.0x02434050 = 0x00000058; # gpio_wan2_pb5: rsvd0, pull-up, tristate-enable, input-enable
123. pinmux.0x02434048 = 0x00000000; # gpio_wan3_pb6: rsvd0, tristate-disable, input-disable	323. pinmux.0x02434048 = 0x00000000; # gpio_wan3_pb6: rsvd0, tristate-disable, input-disable
124. pinmux.0x02434040 = 0x00000058; # gpio_wan4_pc0: rsvd0, pull-up, tristate-enable, input-enable	324. pinmux.0x02434040 = 0x00000058; # gpio_wan4_pc0: rsvd0, pull-up, tristate-enable, input-enable
125. pinmux.0x02431018 = 0x00000058; # gpio_aud0_pj5: rsvd0, pull-up, tristate-enable, input-enable	325. pinmux.0x02431018 = 0x00000058; # gpio_aud0_pj5: rsvd0, pull-up, tristate-enable, input-enable
126. pinmux.0x02431010 = 0x00000000; # gpio_aud1_pj6: rsvd0, tristate-disable, input-disable	326. pinmux.0x02431010 = 0x00000000; # gpio_aud1_pj6: rsvd0, tristate-disable, input-disable
127. pinmux.0x02432028 = 0x00000405a; # dmic4_clk_pm5: rsvd2, pull-up, tristate-enable, input-enable	327. pinmux.0x02432028 = 0x00000405a; # dmic4_clk_pm5: rsvd2, pull-up, tristate-enable, input-enable
128. pinmux.0x02432020 = 0x000004002; # dmic4_dat_pm4: rsvd2, tristate-disable, input-disable	328. pinmux.0x02432020 = 0x000004002; # dmic4_dat_pm4: rsvd2, tristate-disable, input-disable
129. pinmux.0x02433020 = 0x000004058; # gpio_pq4_pi4: rsvd0, pull-up, tristate-enable, input-enable	329. pinmux.0x02433020 = 0x000004058; # gpio_pq4_pi4: rsvd0, pull-up, tristate-enable, input-enable
130. pinmux.0x02433028 = 0x000004000; # gpio_pq5_pi5: rsvd0, tristate-disable, input-disable	330. pinmux.0x02433028 = 0x000004000; # gpio_pq5_pi5: rsvd0, tristate-disable, input-disable
131. pinmux.0x02433030 = 0x000004058; # gpio_pq6_pi6: rsvd0, pull-up, tristate-enable, input-enable	331. pinmux.0x02433030 = 0x000004058; # gpio_pq6_pi6: rsvd0, pull-up, tristate-enable, input-enable
132. pinmux.0x02433038 = 0x000004054; # gpio_pq7_pi7: rsvd0, pull-down, tristate-enable, input-enable	332. pinmux.0x02433038 = 0x000004054; # gpio_pq7_pi7: rsvd0, pull-down, tristate-enable, input-enable
133. pinmux.0x02430020 = 0x00000001; # gpio_cam1_pn0: rsvd1, tristate-disable, input-disable	333. pinmux.0x02430020 = 0x00000001; # gpio_cam1_pn0: rsvd1, tristate-disable, input-disable
134. pinmux.0x0c301000 = 0x00000058; # gpio_sw1_pff1: rsvd0, pull-up, tristate-enable, input-enable	334. pinmux.0x0c301000 = 0x00000048; # gpio_sw1_pff1: rsvd0, pull-up, tristate-disable, input-enable
135. pinmux.0x0c301008 = 0x00000058; # gpio_sw2_pff2: rsvd0, pull-up, tristate-enable, input-enable	335. pinmux.0x0c301008 = 0x00000058; # gpio_sw2_pff2: rsvd0, pull-up, tristate-enable, input-enable
136. pinmux.0x0c301010 = 0x00000058; # gpio_sw3_pff3: rsvd0, pull-up, tristate-enable, input-enable	336. pinmux.0x0c301010 = 0x00000058; # gpio_sw3_pff3: rsvd0, pull-up, tristate-enable, input-enable
137. pinmux.0x0c301018 = 0x00000058; # gpio_sw4_pff4: rsvd0, pull-up, tristate-enable, input-enable	337. pinmux.0x0c301018 = 0x00000058; # gpio_sw4_pff4: rsvd0, pull-up, tristate-enable, input-enable

Figure 4.3.5 Text difference between the default CFG (left) and the CFG we created (right)

Then we flashed the new CFG file to board and used devmem2 to check the value at the location. The command used is shown in Figure 4.3.6.

```
ubuntu@tegra-ubuntu:~$ sudo ./devmem2 0x0c301000
/dev/mem opened.
Memory mapped at address 0x7f8fcbc000.
Value at address 0x0C301000 (0x7f8fcbc000): 0x00000048
```

Figure 4.3.6 Running devmem2 on the location

The value displayed was 0x48 instead of 0x58 this time. The 4th bit was a 0, meaning the tri-state is passthrough, or in other words, disabled. Before we flashed the new CFG file, the value was 0x58 where the 4th bit was a 1, so the tri-state was enabled before. Note that the bit counting usually starts at 0, so the 4th bit is actually the 5th binary digit.

4.4 Process

A. Planned Schedule

We came up with an eight-week plan after we arrived at NVIDIA, shown in Figures 4.4.1 and 4.4.2. In the first two weeks, we would be doing background research on the project, including getting familiar with the working environment, reading the internal Wiki pages, and trying to flash the board. Then we would decide what features we would actually implement in tool, and came up with a GUI Mockup as well as a logic diagram.

Coding would start on the fourth week, after we had the design reviewed. We planned to finish the coding and testing in the three weeks after. We would divide the work into the front-end and back-end. We would also write documentation files for our tool, and there would be separate documentation for the future users and potential developers. The final MQP report would be finished in the last two weeks. The code and documentation would be completed in the last week.

We planned to have weekly meetings with Professor Claypool and our mentor Christian Gonzalez. In the meeting, we would discuss the past week's accomplishments, plans for the upcoming week- and record his feedback. This way we could keep our supervisors updated with our process, and receive comments on any problems. Stephen was also very responsive through emails or Communicator about his requirements for our tool.

Task Name	Start	Finish	Duration
TX2 GUI Tool Project			
[-] Planning	01/10/17	01/25/17	11.25d
Project Kickoff	01/10/17	01/11/17	2d
[-] Project Research	01/11/17	01/18/17	5.75d
Read Wikis/Quick Start Guides	01/11/17	01/12/17	2d
Modify Spreadsheet	01/13/17	01/18/17	3.25d
Flash CFG with small pin change/pad voltage	01/17/17	01/18/17	1.75d
[-] Feature Breakdown	01/20/17	01/25/17	3.25d
Create list of Features (core functionality)	01/20/17	01/23/17	2d
Stephen Review	01/24/17	01/25/17	1.25d
[-] Design	01/24/17	01/27/17	4d
Start Design	01/24/17	01/26/17	2.75d
Final Mockups Review	01/26/17	01/27/17	1.25d
[-] Development	01/30/17	02/23/17	18.75d
Backend Server	01/30/17	02/17/17	15d
Front end	01/30/17	02/17/17	15d
Scripts	01/30/17	02/17/17	15d
[-] Defect Review	02/20/17	02/23/17	3.75d
Bugfixing	02/20/17	02/21/17	2d
QA	02/22/17	02/23/17	1.75d
Documentation	02/23/17	02/24/17	2d
Finish Final Project	02/23/17	02/24/17	1.25d
[-] Final Report	02/27/17	03/03/17	5d
Write	02/27/17	03/01/17	2.75d
Finish Report	03/01/17	03/03/17	2.25d

Figure 4.4.1 Planned Schedule View

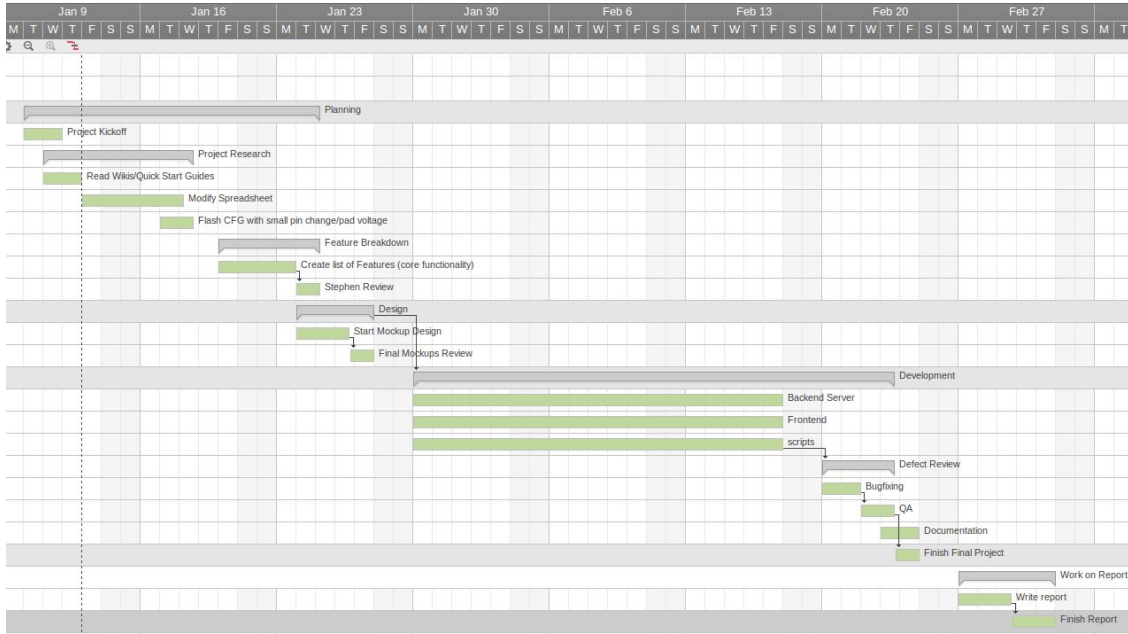


Figure 4.4.2 Gantt Chart of the Planned Schedule

B. Tools

We have used many useful tools throughout the project to keep our work organized. One that we used the most often is called Trello, as shown in Figure 4.4.3. Tasks were broken down into small pieces each week, and each task was assigned to a team member. A task would show up in the “Backlog” section first, and then be moved to “Current” when it was being worked on. Then it was moved to “Review” to be reviewed by the other members, and finally moved to “Done”. We also kept track of open and fixed bugs on the board. This system worked well to alert other members of what has been completed or still needs to be done. Trello function as a scrum board for us.

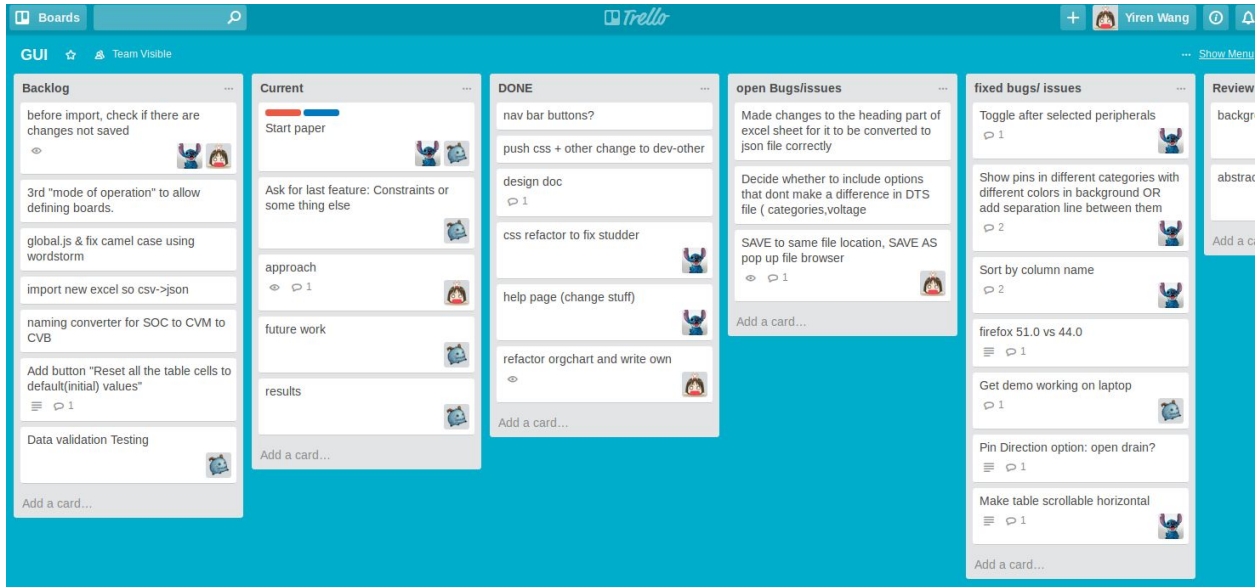


Figure 4.4.3 Trello Board View

Another tool that we used is the Gerrit Code Review website. NVIDIA uses an internal instance of Gerrit where only NVIDIA developers merge and review code. Our team created a new Git branch off of a NVIDIA repository for our only project. Any code that was completed was tested locally then pushed to Gerrit. This process was done often so all members had the most updated code. After we felt the each code portion was complete, we would start the review process. We invited our mentor and NVIDIA JavaScript reviewers to check our code, and received useful feedback regarding any code styling problems. Through this process we learned to write code in a professional style consistent with NVIDIA's standards and include detailed comments.

4.5 Design

A. GUI Mockup

To begin the design process we created a GUI Mockup which can be found in Appendix D. Considering the designs of GUI tools from other sources and our project requirements, we decided to do two tabs for our first draft. The main page would have a table similar to the pinmux spreadsheet. This would mimic the spreadsheet functionality and give users who are used to the old design the same idea. The sidebar would have another list of all the categories which would filter the pinmux table. On the initial webpage load, the table will be filled with all pins, and the sidebar categories are all selected. The users can then select or deselect categories from the side to change what is filled in the table. The export and flash buttons would also be on this page which give users the ability to export to CFG files.

The second tab is a connector tree which starts with the base SoC (we defined the SoC to be the TX2). The users then use the sidebar palette to drag in boards to add to the SoC. Based on what the user has added on the device tree, the pinmux table on the first tab will change. As more connectors are added, more constraints are applied to the spreadsheet. The user can also drag a connector back to the sidebar to remove it from the tree.

Once the mockup was completed a bootstrap template acted as the starting point and was slightly modified to follow the mockup. After a few weeks the mockup helped guide the design but as with initial mockups, features would be added or removed along the development process. For example, comments from our team led to removing the flash button because users should know what to do with a CFG file. Save and import buttons were also added to give users the

ability to save their current changes from the table and import them later. Another tab was added to change voltages in its own separate table instead of the spreadsheet table. Comments from our team helped make sure this tool was complete with all the mandatory features.

B. Logic Diagram

A box diagram was created to show the logic behind the project, which can be seen in Appendix C. The diagram splits the client and server into two big boxes. The client side refers to the logic on the web pages, and the server side describes the process running on the Node.js back end when requests are received from the client.

The client side starts with opening the webpage. A loading page request is sent to the server, and the server reads from the board definition data files and constructs HTML pages to return to the client.

After the pages are loaded, the user can interact with the three buttons on the navigation bar. The “Export File” button is used to save the user’s configurations on the web pages, and will send an AJAX “save” request to the server. The server generates a temporary configuration file in the TXT format in the backend, and returns a download link to the client. The browser will download the file and the user can save it on the computer. The “Import File” button can read from any temporary configuration file and restore the user’s configurations on the web page.

The “Generate CFG” button generates CFG files based on the configurations from the web pages. It sends an AJAX “build” request to the server along with the pinmux configuration information. The server will generate three DTS files based on the information, and then creates a Python process, which reads in the DTS files, existing Python script and some helper TXT

files, to create the CFG files. In the end, the server archives all the CFG files it has generated into a ZIP file, and returns the download link to the client for downloading the ZIP file.

The user can also modify the pin configurations on the web pages before pressing any of these buttons. On the default Pinmux page, users can filter the pin table using the category filter, or change the pin configurations in the table. The table will show errors in red if any of the pins are configured incorrectly. By clicking on the three tabs on the top right corner, the user can switch to the Connections page or the Pad Voltages page.

The Connections page is used to select the current board state. The user can change the state by dragging a board to the chart, or removing a board from the chart. The most recent board dragged into the chart will become the current board state. The change of the board state will be reflected on the Pinmux page, because some pins will not be configurable if the chart has a CVM or CVB board on it. The Pad Voltages page is used to select voltages for each pad. The modified voltages will also be displayed in text on the Pinmux page. When the board state changes, some default pad voltages will also change according to the board definition files.

4.6 Coding

A. Development

The Tegra Pinmux GUI Tool was developed through weekly iterations and was improved based on feedback. As a starting point in development, the first step was to emulate the spreadsheet on our web app. We started off with a template page with a table and changed around some of the field names to be more appropriate. Next, we used a CSV to JSON converter that retained all the dropdowns in the Excel spreadsheets. The Excel spreadsheet could not be instantly copied and pasted in, as some columns had different widths that would bug the converter. The detailed instructions on how to cleanse the Excel spreadsheet for JSON conversion can be found in Appendix I.

After successfully producing a JSON file, we setup our Node.js server to host our simple template website and send that JSON file when requested. The next part of our development was to create logic that would populate the table given the JSON object. Many fields were one-to-one, so a table row value could directly map to a table column name. Through the use of some string templating to create HTML select tags, we maintained the dropdown menus. When we reached this point, we ran into a problem where users changed their pinmux settings and wanted to export, our app had to run through the whole table and extract all the selected table values. We didn't like the process of parsing through HTML text, as it seemed error prone because it would be string dependent, so we created a global variable called `pinDictionary[]` that mapped unique IDs to a pin object. A pin object has fields that are all the column names in the Excel sheet, and has values that are the ones the user selected.

It was not clear at first how the Excel spreadsheet generated DTS files, but after consulting with Richard Shen, we found Visual Basic scripts that the spreadsheet had used. We decided to offload the DTS and CFG file generation process to the server because the client side should focus on UI changes. Note that Node.js runs JavaScript, so the client and server can interchange functions if needed.

Reaching a point with intermediate DTS files, our tool needed to automate what users would manually do at this point. Normally, users would transfer the file over to Linux, run DOS2UNIX to remove line endings, and then run the DTS2CFG python script. The script needs address info text documents as well as the DTS files for input. The address info text documents provide a mapping from pin names to address locations (similar to the TRM mapping found in Section 4.3). We developed our server side to spin up another child process to have Python execute the script (Figure 4.6.1).

```
python = require('child_process').spawn('python',  
  [python_tool_path + '/pinmux-dts2cfg.py',  
    '--pinmux',  
    python_tool_path + '/addr_info.txt',  
    python_tool_path + '/gpio_addr_info.txt',  
    python_tool_path + '/por_val.txt',  
    dts_full_path + pinmux_dts,  
    dts_full_path + gpio_dts,  
    '1.0']
```

Figure 4.6.1 Node.js code that spawns a child process
and runs the Python script with specified arguments

To reiterate, our tool does the same CFG file generation process as users had to do with a spreadsheet before, but this tool hides and automates the process in the backend. When the two CFG files are created from the Python script - one for pinmux configuration, another for pad

voltage configuration, they are zipped up using the Node module, Archiver. The server then alerts the client with the zip file location, so the user can download it. We were almost done with developing the spreadsheet tab, but adding pad voltages needed some UI design considerations.

A small but important tab is the pad voltage tab. While initially located above every grouping in the spreadsheet (Figure 4.6.2), feedback we received leaned us towards creating a separate tab in the GUI tool. Adding a separate column for IO Block Voltage in the spreadsheet table made all other columns tighter. The overall look and feel was more important, so we decided to to put the pad voltages in another tab. The pad voltages tab contains just two columns, Pad Group Name and IO Block Voltage, as seen in Figure 4.6.3.

Customer Usage	Pin Direction	Req. Initial State	Wake Pin	Lock	3.3V Tolerance Enable	LPDR Enable	EQOS LPBK Enable	Ext Pull Up Value (Ω)	Ext Pull Down Value (Ω)	Req. Deep Sleep State	IO Block Voltage	Customer Usage Description or Net Names
UD3_TXD	Output					Disable					1.8V	UART4_TX (CVM: UART3_TX)
UD3_RXD	Input	Int PU				Disable						UART4_RX (CVM: UART3_RX)
UD3_RTS	Output					Disable						UART4_RTS (CVM: UART3_RTS)
UD3_CTS	Input	Int PU	No			Disable						UART4_CTS (CVM: UART3_CTS)
GPIO3_PB.04	Output	Drive 1				Disable						GPIO (CVB_BT_EN)

Figure 4.6.2 Excel spreadsheet showing IO Block Voltage on individual row

Pad Name	Voltage
CONN	1.8V
AUDIO	1.8V
AUDIO_HV (3.3V Capable)	3.3V
DMIC_HV	1.8V
CAM	1.8V
PEX_CTL	1.8V
SDMMC1_HV (3.3V Capable)	1.8V/3.3V
SDMMC2_HV (3.3V Capable)	3.3V
SDMMC3_HV (3.3V Capable)	1.8V/3.3V
SDMMC4	1.8V
SYS	1.8V
UART	1.8V

Figure 4.6.3 Pad Voltages tab in the Pinmux GUI Tool

The last tab we developed is the Connections tab. This tab is an additional feature that is not native to the spreadsheet. The purpose of the Connections tab is to allow users to define their own boards and the constraints. In our implementation, the Jetson module (CVM) and Jetson board (CVB) are defined. Other boards could be created, but we hadn't done so. A user story of this tab is if a user just has the Tegra Chip and wants full flexibility in pinmux options. If only the Tegra SoC is selected in the connections tab, then all pins in the spreadsheet view are fully configurable. When the user adds the CVM to the connection chart, some pins are going to be constrained because in order for the CVM to function, some pins should be set to specific functions. Lastly, when the user wants to add his board to the connections design, assuming he has created a data file for his board or is using the default CVB, the constraints of the board would also be applied. At this point, the user should only be able to pinmux GPIO expansion

header pins, or the pins of IO Controllers they are not planning on using. An example is if a user was not going to use the SD Card slot, then they can pinmux it for other functions. Our implementation does not allow for creating data files in the application, but the CVM and CVB data files could be modified to add or remove constraints in the JSON object.

Throughout the project, aside from feedback about features, we had feedback on our code as well. We refactored our code to match the spacing and naming convention at NVIDIA. We submitted patch sets to encapsulated global variables. More refactoring could have been done, but due to unfamiliarity of some frameworks (See Future Work), we chose to skip the potential for more abstracted code in favor of good documentation and feature development.

B. Third-Party Frameworks and Modules

To help meet the requirements of our project without reinventing the wheel, we used third party frameworks and modules. The three frameworks that were used are jQuery, jQuery UI, and Bootstrap. jQuery let us quickly grab HTML documents, easily add event listeners, and asynchronously fetch files from the server in few lines of code. jQuery UI was essential to implement drag and drop functions on the connections page. jQuery UI easily let us add `draggable()` and `droppable()` listeners to the right UI elements. In the code fragment below (Figure 4.6.4), we set `draggable` and `droppable` event listeners on the board and sidebars for the connections tab.

```

99 /**
100 Set the draggable and droppable listeners for the chart elements
101 */
102 function setChartListeners() {
103
104     // Set each board to be draggable
105     $(".node").draggable({
106         snap: ".node",
107         snapMode: "outer",
108         revert: true
109     });
110
111     // Set the navigation bar to receive the boards dropped
112     $(".sidebar").droppable({
113         accept: ".node",
114         drop: function(ev, ui) {
115             // Get the dragged item's text
116             var droppedId = ui.draggable[0].id;
117
118             // The board should be the latest board added
119             if (globals.deviceTree.length-1 !== Number(droppedId)) {
120                 alert("Please remove the boards at the bottom first.");
121             } else if (globals.deviceTree.length === 1) {
122                 alert("The chart cannot be empty.");
123             } else {
124                 // Update the device tree
125                 console.log('Removing the board ' + globals.deviceTree[globals.deviceTree.length-1] + '...');
126                 globals.deviceTree.pop();
127                 globals.connectionsBoardState = globals.deviceTree[globals.deviceTree.length-1];
128
129                 // Redraw the chart with new device tree
130                 drawBoards();
131             }
132         }
133     });
134 }

```

Figure 4.6.4 JavaScript code to set drag and drop listeners on a board and sidebar using jQuery UI

The Bootstrap framework was essential to give our webpage a nice look and feel. Aside from the sample template we started with, Bootstrap also allows the responsive tables - allowing us to resize the window, but still have the table be scrollable.

We added various third party Node modules to our server side code. The first node modules is Commander. Commander allows us to parse command line arguments and respond with appropriate logic. We used another Node module called Archiver to zip up the generated CFG files and provide it as one file to download for the user. Express is a Node module used to serve static web files. It is always listening for file requests from a client and will respond with

the file if it exists. The last Node module we use is Electron. Electron is for our final stage packaging of our web app. It builds a desktop app so the user no longer needs to worry about starting up with the correct command line arguments; Electron makes our app just click and start. Below in Figure 4.6.5, our application is an Electron app. Figure 4.6.6 shows the various distributables for Windows, Linux, and Mac OS.

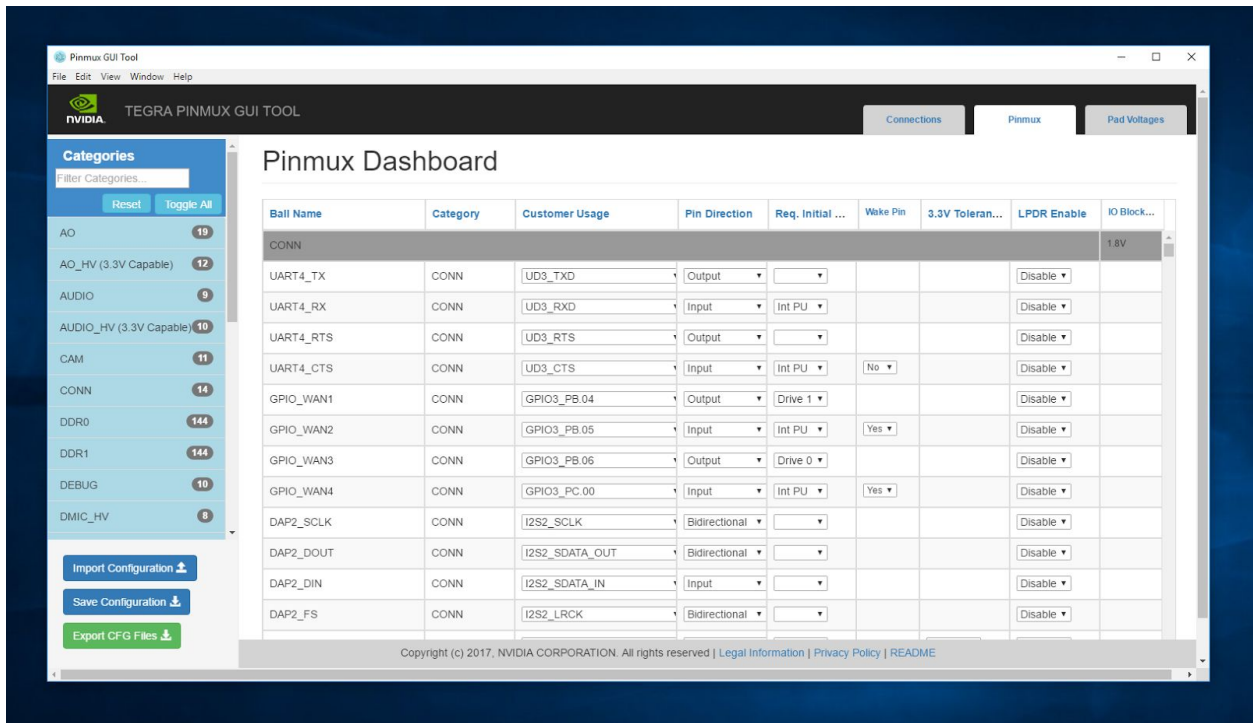


Figure 4.6.5 Our Pinmux GUI tool deployed as an Electron app

Name	Date modified	Type	Size
RELEASES	2/28/2017 7:23 PM	File	1 KB
Setup.exe	2/28/2017 7:23 PM	Application	57,033 KB
Setup.msi	2/28/2017 7:23 PM	Windows Installer ...	57,052 KB
tegra_pinmux_gui_tool-1.0.0-full.nupkg	2/28/2017 7:23 PM	NUPKG File	56,861 KB

Figure 4.6.6 (A) Our Pinmux GUI tool as an Electron app for Windows

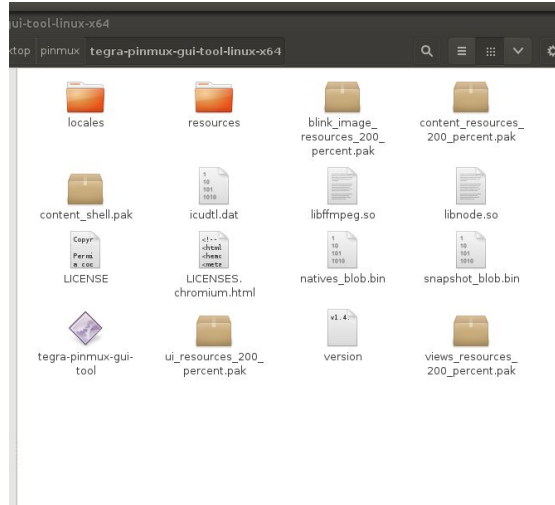


Figure 4.6.6 (B) Our Pinmux GUI tool as an Electron app for Linux

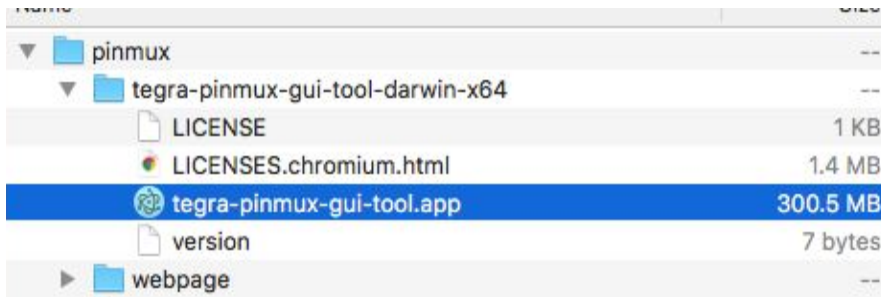


Figure 4.6.6 (C) Our Pinmux GUI tool as an Electron app for Mac OS

The complete list of third party libraries we used along with their copyright information can be found in Appendix B.

5 Results

Results talks about developed features in detail, and how the feature satisfies requirements stated earlier. Some features do not satisfy a requirement, but do benefit the user in terms of usability.

5.1 Developed Features

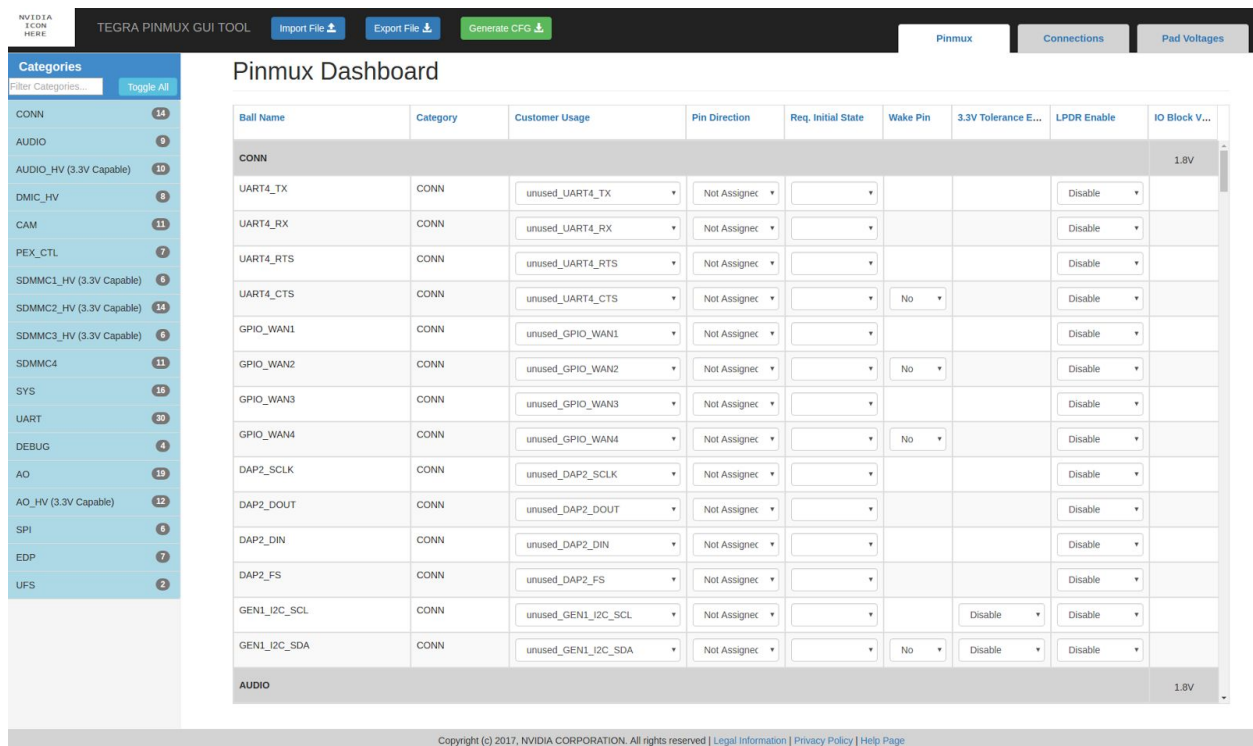


Figure 5.1.1 Tegra Pinmux GUI Tool (Pinmux Tab)

The initial screen when loading the pinmux web app features a navigation bar and 3 tabs on the top right corner, as shown in Figure 5.1.1. The Pinmux Tab is shown by default and lists all the data imported from the previous Excel spreadsheet. Each pin can be customized to the user's desired configuration. All changes made to the table are checked for data validation.

Therefore the user cannot specify a configuration that is not possible, such as setting a pin direction for a pin that is unused. If the user does make a mistake, a window will pop up showing the error, and the table will display a red border around the data. An example of the data validation function is shown in Figure 5.1.2.

Ball Name	Category	Customer Usage	Pin Direction
CONN			
UART4_TX	CONN	unused_UART4_TX ▼	Input ▼

Figure 5.1.2 Pinmux Tab Data Validation Example

After the user completes all the necessary changes, they are able to export the CFG files directly, which is a feature that the previous spreadsheet does not allow. If the user wants to save his or her current work and return later, we also created a feature that saves all the work to a text file. The user may import the file and achieve the saved data, which was one of the application’s requirements. The buttons for these features are in the navigation bar, as shown in Figure 5.1.3.

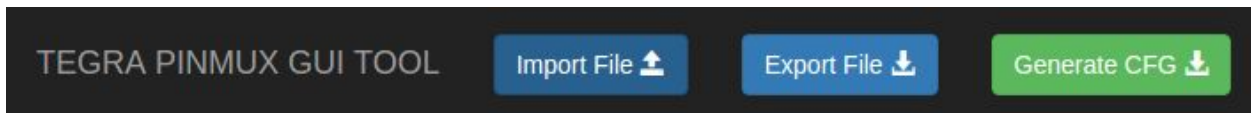


Figure 5.1.3 Navigation Bar Buttons

One flaw of the old spreadsheet configuration was that there were over 600 pins listed while many were not configurable. We sought out to minimize the amount of pins our tool displays by only showing the pins that the user can change. This reduced the amount of data by half so our table is smaller than the spreadsheet.

Another feature developed to help minimize the large table of data was the Category (Pad Group) Filtering. All pins are grouped by pad groups so the user may choose to only display specific groups by selecting them on the left sidebar. The category filtering sidebar can be seen in Figure 5.1.4. The toggle button toggles all of the pin groups selected or unselected. The user may also write in the filter bar to filter by name. Any pin group selected or unselected in this list changes what is visible in the main pinmux table. These filtering capabilities were not a requirement of the app, but help to avoid cluttering the table.



Figure 5.1.4 Pinmux Tab Pad Group Filtering

This Connections Tab (second tab) displays a draggable and droppable tree so the users may add and remove boards to be connected, which in turn constrains the amount of pins that may be configured on the Pinmux Tab. An important feature of the table is to only allow pin configurations that are possible, and constrain some pins depending on what the user has

physically connected to their personal board. This partially completes the requirement of constraints based on connected boards. In our application, constraints would prevent a whole row from being modified. In practice, constraints should allow some dropdown elements to be unselectable, not disabling the whole row. Figure 5.1.5 shows the Pinmux Tab with some pins constrained.

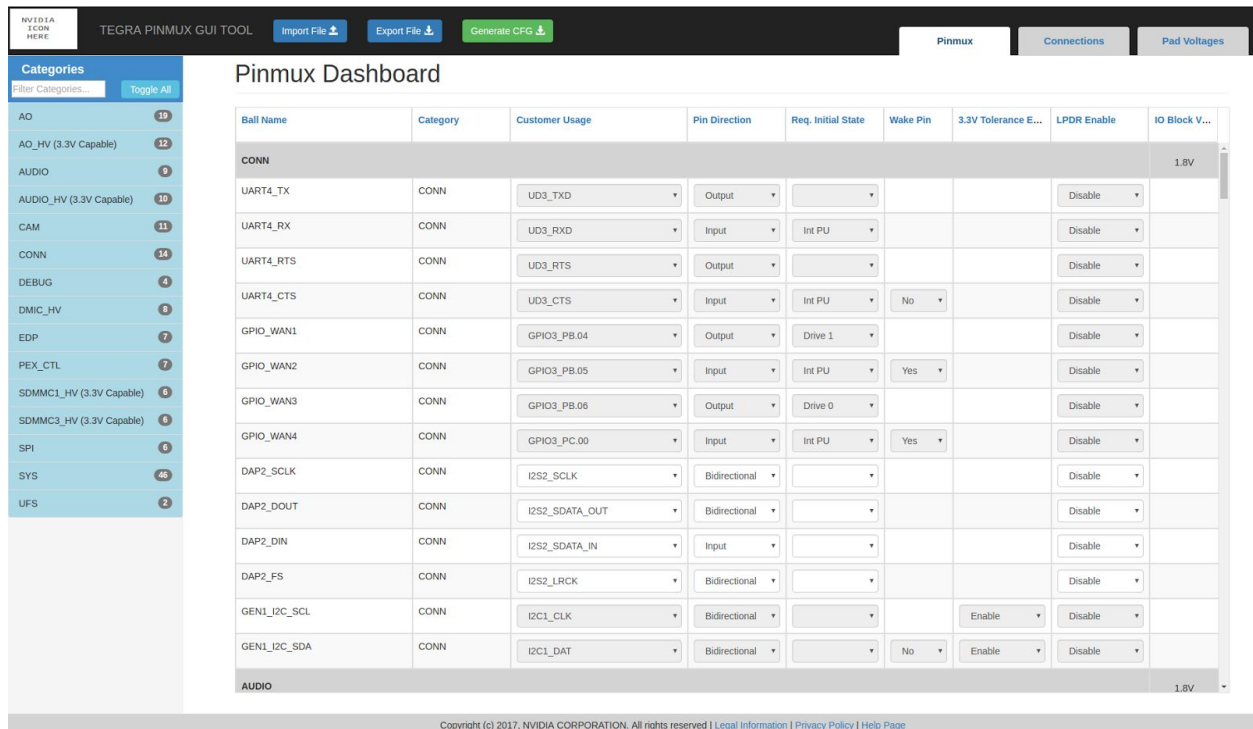


Figure 5.1.5 Pinmux Tab with Pins Constrained

Our third tab, Pad Voltages, is similar to our pin configuration table, except that it only contains the pad group names rather than all the pins. This is because a pad voltage is selected for the entire group rather than each individual pin. It was better to display this in its own page and table rather than combining it with the pin configuration table to clearly show that this is a group option. However, the pad voltages are still displayed in text in the table for user reference.

5.2 Evaluation

The evaluation process can be broken down into two parts: objective testing and subjective testing. Objective testing is checking the CFG files output by our tool and comparing them with the pre-existing spreadsheet's output. If the two CFG files match, then our tool will flash the same values to a Jetson Board as a spreadsheet would (as shown in Section 6.3). The result is binary: a file match, or mismatch. Our process of objectively testing our code did actually catch errors that we would have otherwise missed. The reason for the discrepancy between our CFG files and the spreadsheet's CFG files was because of unnoticed edits on the spreadsheet that caused it to stray from its default values. The initial state between the GUI tool and the spreadsheet must be the same if the output is to stay the same. After we double checked our initial state of our GUI tool, we tested and found all CFG files generated match.

The other component of evaluation is the subjective testing. This form of testing is based on feedback from our peers: Stephen Warren, Christian Gonzalez, and the L4T team. We want to make sure that we have built the functionalities that they required, and the look and feel of the tool meet their expectations. We also go over our backend design and see if adding additional requirements would require code refactoring. For example, after we had demonstrated the working Pinmux Tab, feedback we received asked for the next feature to be a Connections Tab that would constrain the spreadsheet based on what boards the user connects together. Examples of feedback that required refactoring code are the patch sets we submit to Gerrit after code review from our peers. If a file had inconsistent spacing, or unnecessary whitelines, or unclear

comments, we would see a comment about it on Gerrit, and push a patchset to fix it. This form of testing is synonymous with the agile development method.

5.3 Summary

At the conclusion of our development cycle, we have a tangible list of features that satisfy all the previously defined requirements. While some features did not satisfy a specific requirement, they still have a beneficial impact for the user.

Features that satisfy a specific requirement:

- Pinmux tab that shows a table where user can select pinmux options
- Connections tab that allows users to drag and drop connectors that apply constraints on the pinmux
- Read from SoC and Board definition files
- Validation checks that display a red border around erroneous fields
- Pad voltage tab to set voltages for pad groups
- Import and Export current configurations
- Generate CFG files of current configuration
- Packaged application as distributables for Mac, Windows, and Linux

Features that were not required but benefit user:

- Pad Group filtering
- Badge icons displaying the number of pins per pad group

More screenshots of the final Tegra Pinmux GUI Tool can be found in Appendix H.

6 Conclusion

NVIDIA produces system on chips such as the Tegra TK1 and TX1. The SoCs are used in modern hardware such as the Nintendo Switch, NVIDIA Shield, and the Google Pixel C. NVIDIA also manufactures development boards, called Jetson Boards, which feature the Tegra SoC. Jetson Boards have pins that connect to peripherals, and most importantly, to the Tegra SoC. NVIDIA allows software to control the functionality of pins on the Jetson Boards. This process is called pinmuxing. Pins can generally be pinmuxed to be the default value, GPIO, or unused.

Developers using the Jetson board would want to pinmux when they are not using a peripheral and want that pin to do other logic. In the past, developers had to modify a Microsoft Excel spreadsheet to select their pinmux options, and then export to an intermediate DTS file. The DTS would then be input to a Python script which converts it to a flashable CFG file. This process is not only Windows OS reliant, but also included additional steps with a DTS file. The main goal of our project is to develop a web application to configure pins and voltages on Tegra powered boards. Requirements of this application were drafted through feedback from NVIDIA internals. The web tool needed to allow usage across multiple platforms. Internal backend design should abstract users from the intermediate DTS file. Constraints should be applied when users add boards to a device tree. Import and export buttons should allow users to save and resume work. The application should also be distributable to users across multiple platforms.

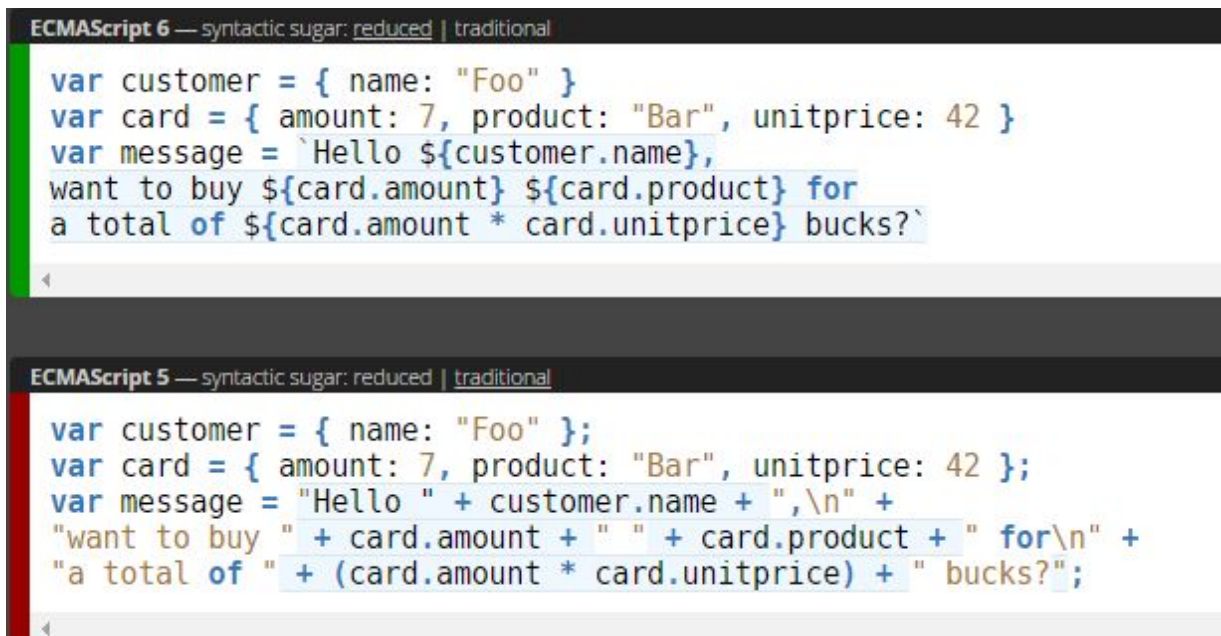
The process we took in completing this project involved researching background information, familiarizing ourselves with development tools, implementing feedback from

weekly iterations of development, and testing and packing the final product. Background research included learning about what a Tegra SoC is, what Jetson boards are, and what pinmuxing is. We used Gerrit and Trello as tools to keep our development process standardized. New features were often added or old features modified a bit after presenting our weeks development to many people. Testing was on the CFG files generated from our tool compared with an Excel spreadsheet. The app was finally packaged and distributable for multiple OSs.

Developed features include a web application with a navigation bar and three tabs. The navigation bar has three buttons for users to import/export their work, and generate CFG files. Three tabs are used for Tegra configuration options. The default tab is the spreadsheet view containing drop down options for pin configurations. The second tab is the connections tab where users drag and drop boards to a device tree. On the last tab, users can set pad voltages for pad groups. The web app was exported as a desktop application for multiple platforms through Electron. Overall, the app has potential to give users and NVIDIA developers a seamless way to configure Tegra settings.

7 Future Work

Due to the constraint of seven weeks, there are many improvements that could help maintain the readability, scalability, and functionality of the Tegra Pinmux GUI Tool but were not implemented. For readability, we could use JavaScript frameworks such as React or Angular to create data models and data bindings for our tables, categories, footers, sidebars, tabs, etc. This would help future developers more easily see what JavaScript code is affecting which UI element. Adoption of ES6 will also help readability, most notably because of its support for string templating. In the current code, we use JavaScript to set the innerHTML of elements to dynamically created strings. The dynamically created strings would look different with the adoption of ES6. On ES6-features.org, an example of the difference in string templating between ES6 and ES5 (our version) is given in Figure 7.1.1:



```
ECMAScript 6 — syntactic sugar: reduced | traditional
var customer = { name: "Foo" }
var card = { amount: 7, product: "Bar", unitprice: 42 }
var message = `Hello ${customer.name},
want to buy ${card.amount} ${card.product} for
a total of ${card.amount * card.unitprice} bucks?`

ECMAScript 5 — syntactic sugar: reduced | traditional
var customer = { name: "Foo" };
var card = { amount: 7, product: "Bar", unitprice: 42 };
var message = "Hello " + customer.name + ",\n" +
"want to buy " + card.amount + " " + card.product + " for\n" +
"a total of " + (card.amount * card.unitprice) + " bucks?";
```

Figure 7.1.1 Difference between ES6 string templating and our current code (ES5)

In terms of scalability, the project could be enhanced by refactoring functions and CSS files. The purpose of many JavaScript functions should be purely functional and avoid touching UI. This will allow for easier unit testing on the functions because testing UI is difficult and hard to measure. Throughout our development, we used AJAX to replace the body of the main index.html page when a user switches tabs, instead of going to a new page in order to preserve global variables. Each tab has different styling so we wrote a function to replace the previous tab's stylesheet with the current tab's stylesheet. Replacing the stylesheets every time a user switched tabs caused stutters, whose lag time was browser dependent. The reason for the stutter was because HTML elements were being shown before CSS styling could be applied to it. In our last week, we managed to create one CSS file for all tabs, but some elements on a specific tab had started behaving strangely. For example, when resizing a window on the spreadsheet tab, the table would start jumping into the category section. Future improvements would create a collective but functional CSS file for the whole website.

Last but not least, additional functionality improvements can come directly from Stephen Warren's wiki. We had scaled down his requirements for the purposes of defining a feasible MQP project. Functionality that we hope to include in the GUI tool include enabling/disabling IO Controllers and auxiliary CPUs, and configuring of DVFS policy. Each of these three functionalities would probably be their own separate tabs, and the whole GUI tool would be a central hub for all the configuration of a Tegra chip and user defined boards.

8 References

AnOldGreenHorn. "Embedding Python In Your C++ Application." *CodeProject*, 22 May 2006, www.codeproject.com/articles/14192/embedding-python-in-your-c-application.

Benchoff, Brian. "The Nvidia Jetson TX1: It's Not For Everybody, But It Is Very Cool." Hackaday. N.p., 24 Nov. 2015. Web. 28 Feb. 2017. <http://hackaday.com/2015/11/24/the-nvidia-jetson-tx1-its-not-for-everybody-but-it-is-very-cool/>.

"Build Desktop Applications." *AppJS*, appjs.com/. Accessed 25 Feb 2017

"Call and Receive Output from Python Script in Java?" *Stack Overflow*, Stack Exchange Inc, 10 Apr. 2012, stackoverflow.com/questions/10097491/call-and-receive-output-from-python-script-in-java.

"Electron." *Electron*, electron.atom.io/. Accessed 26 Feb 2017

"Embedded Systems Development Solutions from NVIDIA Jetson." *Development Solutions from NVIDIA Jetson | NVIDIA*, NVIDIA, www.nvidia.com/object/embedded-systems.html.

"Embedding Python in Another Application." *Python Documentation*, docs.python.org/2/extending/embedding.html.

Extrabacon. "Extrabacon/Python-Shell." *GitHub*, GitHub, Inc., 10 Mar. 2016, github.com/extrabacon/python-shell.

Franklin, Dustin. "NVIDIA® Jetson™ TX1 Supercomputer-on-Module Drives Next Wave of Autonomous Machines." Parallel FORALL. NVIDIA, 11 Nov. 2015. Web. 28 Feb. 2017. <https://devblogs.nvidia.com/parallelforall/nvidia-jetson-tx1-supercomputer-on-module-drives-next-wave-of-autonomous-machines/>.

"GUI Programming in Python." *Python Wiki*, 5 Nov. 2016, wiki.python.org/moin/GuiProgramming.

"Introducing NVIDIA® Tegra® 4, The World's Fastest Mobile Processor." Tegra 4 Processors | NVIDIA. NVIDIA, n.d. Web. 28 Feb. 2017.
<<http://www.nvidia.com/object/tegra-4-processor.html>>.

"Introducing The Tegra X1 Super Chip from NVIDIA." *Super Chip | NVIDIA Tegra*, NVIDIA, www.nvidia.com/object/tegra-x1-processor.html.

"Java GUI Frameworks. What to Choose? Swing, SWT, AWT, SwingX, JGoodies, JavaFX, Apache Pivot?" *Stack Overflow*, Stack Exchange Inc, 10 Oct. 2013, stackoverflow.com/questions/7358775/java-gui-frameworks-what-to-choose-swing-swt-awt-swingx-jgoodies-javafx.

"JetPack for L4T." *NVIDIA Developer*, NVIDIA, 23 Nov. 2016, developer.nvidia.com/embedded/jetpack.

"Jetson TX1." Jetson TX1 - ELinux.org. N.p., n.d. Web. 28 Feb. 2017.
<http://elinux.org/Jetson_TX1>.

"Modules." *Python Documentation*, Python Software Foundation, 8 Dec. 2016, docs.python.org/2/tutorial/modules.html.

"Pin Mux Tool." *Pin Mux Tool - TI Software Folder*, Texas Instruments, 16 June 2014, www.ti.com/tool/PINMUXTOOL.

"Pin Mux Tool." *Pin Mux Tool - PINMUXTOOL - TI Software Folder*, NVIDIA, www.ti.com/tool/PINMUXTOOL.

"PINCTRL (PIN CONTROL)." *The Linux Kernel Archives*, NVIDIA, www.kernel.org/doc/Documentation/pinctrl.txt.

"Run a Python Script ." *Stack Overflow*, Stack Exchange Inc, 23 Sept. 2010, stackoverflow.com/questions/3781851/run-a-python-script-from-another-python-script-passing-in-args/.

"Tegra-Pinmux-Scripts." *GitHub*, NVIDIA, 31 Aug. 2016, github.com/NVIDIA/tegra-pinmux-scripts.

9 Appendices

Appendix A Readme.txt

Pinmux GUI Tool

A local web app GUI tool to allow configuration of pinmux and pad voltages on the jetson-tx2 development board.

0. INSTALLATION

NOTE: \$PINMUX_DIR refers to the directory you extracted the pinmux tool to.

Requirements:

- Install NodeJS 6.9.5 on your system:

```
$ curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
```

```
$ sudo apt-get install -y nodejs
```

- One of the following browsers:

Firefox - above 51.0

Chrome - above 56.0.2924.87

- Install required NodeJS modules:

```
$ cd $PINMUX_DIR/pinmux
```

```
#The below command will install the node modules: Archiver 1.3.0, Commander 2.9.0,
```

```
#Express 4.14.1, and Electron 1.4.15.
```

```
$ npm install
```

- The following are used but do not require any installation:

```
jQuery 3.1.1
```

```
jQuery UI 1.12
```

```
Bootstrap 3.3.7
```

Usage:

```
$ cd $PINMUX_DIR/pinmux
```

```
$ npm start
```

Then visit "localhost:8080" on a web browser.

For usage and help:

```
# node http-server.js -h
```

1. IMPORTANT

See validation_rules.txt to see all validation rules used.

2. DESCRIPTION

The tool reads in data files for a Tegra board, and allows users to change the board connections, configure the pinmux settings as well as the pad voltages. The tool can output a zip file containing cfg files that can be flashed onto the MBL.

The tool will apply data validation rules to prevent users from configuring pins incorrectly, and allow constraints - for when a user wants to modify pins pertaining to certain board connections.

The tool allows users to save their board and pinmux configurations by downloading a txt configuration file. They can import that file into the tool later and restore their configurations on the webpage.

The tool seeks to replace the old method of editing spreadsheets on windows, exporting DTS files, moving them to unix, and running some python scripts to convert them to CFG files. Instead it can generate CFG files on any platform.

3. FILE DETAILS

Below is the description of each folder and important files for reference:

dist/* - framework files (jquery, bootstrap)

node_modules/* - modules used by Node.js (express)

css/* - style sheets for webpage

js/* - javascript files used in webpage

validation_rules.txt - validation rules used to check the pinmux configurations

http-server.js - server side code executed by NodeJS

output/ - folder is created once files are generated

 output/cfg - contains generated cfg files

 output/dts - contains generated dts files

 output/save - contains user saved configuration file

python_tool/ - contains the python script tool

 python_tool/pinmux-dts2cfg.py - python script used to convert dts to cfg files

 python_tool/*.txt - helper files used with pinmux-dts2cfg.py

data/* - JSON data files read in to generate the pinmux table

data/pad.txt - defines the pads needed for generating pad.dts

data/soc.txt cvm.txt cvb.txt - different data files to read in

index.html - webpage for client to see when loading localhost:8080

connections_body.html - webpage for the connections page

spreadsheet_body.html - webpage for the spreadsheet page

voltages_body.html - webpage for the pad voltages page

images/icon_placeholder.png - icon for the webpage

Appendix B Third Party Documentation

3rd Party Code: Electron

Where: <http://electron.atom.io/>

Description: Electron is a framework for creating native applications with web technologies.

3rd Party Code: jQuery UI

Where: <http://jqueryui.com/>

Description: jQuery plugin allows elements to be dragged, dropped, and selected.

3rd Party Code: jQuery

Where: <http://jquery.com/>

Description: jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.

3rd Party Code: Node.js

Where: <https://nodejs.org>

Description: Node.js is a JavaScript runtime that uses event-driven, non-blocking IO model that makes it lightweight and efficient.

3rd Party Code: Bootstrap

Where: <http://getbootstrap.com>

Description: Bootstrap is the HTML, CSS, and JS framework for developing the responsive UI.

3rd Party Code: Express (Node Module)

Where: <http://expressjs.com/>

Description: Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

3rd Party Code: Node-Archiver (Node Module)

Where: <https://www.npmjs.com/package/archiver>

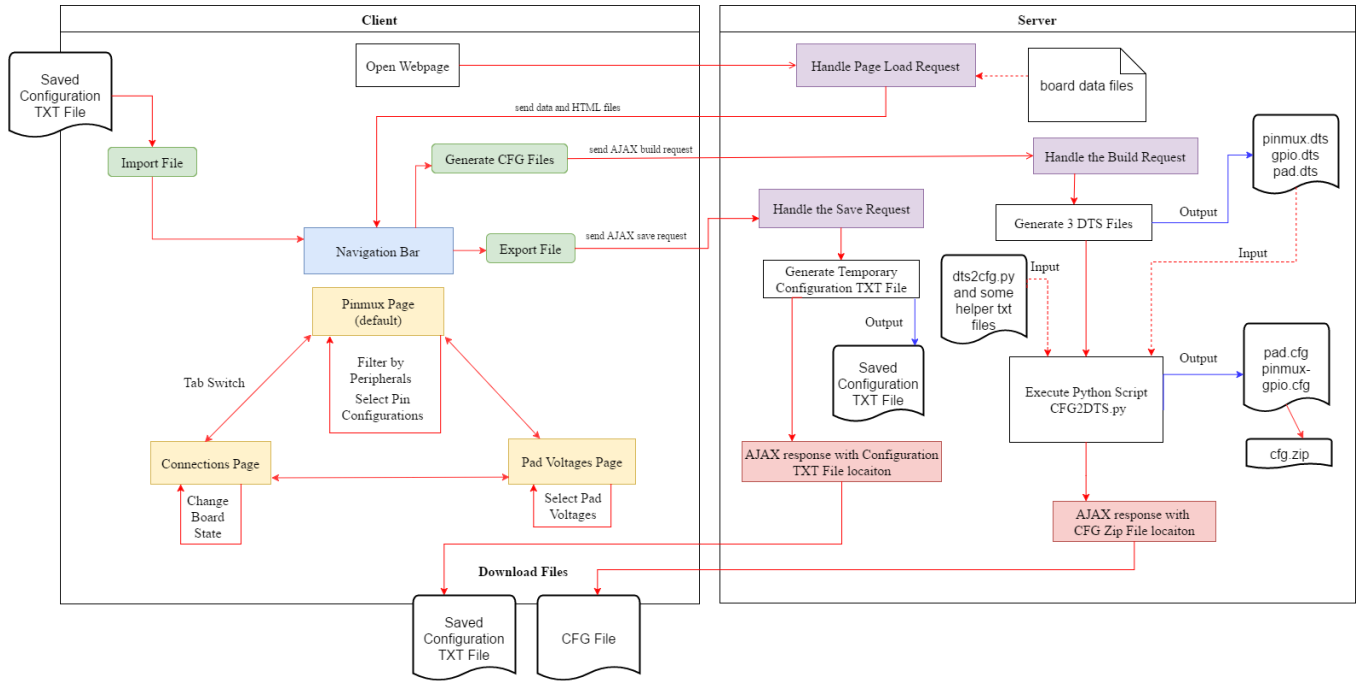
Description: A streaming interface for archive generation. Allows for files to be saved.

3rd Party Code: Commander (Node Module)

Where: <https://www.npmjs.com/package/commander>

Description: The complete solution for node.js command-line interfaces, inspired by Ruby's commander.

Appendix C Box Diagram



Appendix D Mockups

Page 1

https://GUIMockup.io

SPREADSHEET CONNECTOR TREE

Peripherals

- ADC (0)
- CAN (2)**
- I2C (0)
- GPIO (0)
- UART (0)
- EXT1 (5)**
- SD (0)

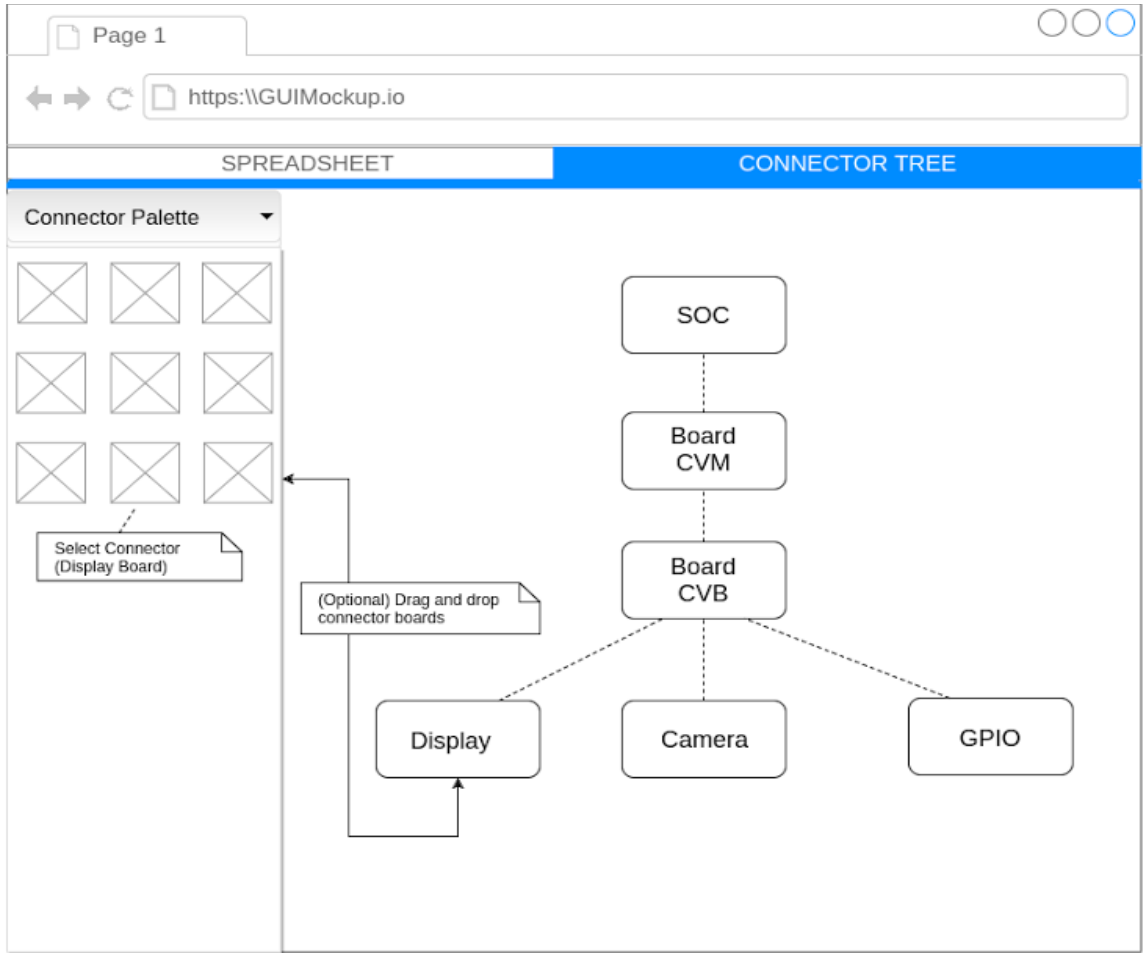
Select

Export CFG file

EXPORT

#	Pin Label	Header	Active	Pin Direction	IO Voltage
1	PA 10	EXT1	<input checked="" type="checkbox"/>	Output	1.8V
2	PA00	EXT2	<input checked="" type="checkbox"/>	Output	
3	PA01	CAN	<input type="checkbox"/>	Output	
4	PA01	CAN	<input type="checkbox"/>	Output	
5	PA01	EX2	<input checked="" type="checkbox"/>	Input	
6	PA01	CAN	<input type="checkbox"/>	Output	
7	PA 10	EXT1	<input checked="" type="checkbox"/>	Output	3.3V
8	PA 10	EXT1	<input checked="" type="checkbox"/>	Output	1.8V
9	PA00	EXT2	<input checked="" type="checkbox"/>	Input	
10	PA01	CLOCK	<input type="checkbox"/>	Output	2.7V

Main page (Alpha)



PINMUX LAYOUT (Beta)

Appendix E Validation Rules

Whenever a user changes any setting, the following validation checks are run:

1. Pin Direction Validation
 - a. If Customer Usage is set to an unused pin, then pin direction MUST be "Not Assigned" or empty

2. Allowed Pin Direction based on selected Customer Usage
 - a. If Pin Direction is "B" then anything works
 - b. If Pin Direction is "O" then only output
 - c. If Pin Direction is "I" then only input

3. Initial State Validation
 - a. If the "Customer Usage" is unused, then they can only select "", "N/A", or "Z"
 - b. Advised when pin direction is "Input" do not use "Drive 0/1"

4. Ext Pull Validation
 - a. if ExtPullUpValue is set or ExtPullDownValue is set, then you cannot select IntPU or Int PD for the required initial state for this pin.

5. Wake Pin Validation

- a. Wake pin cannot be "yes" if a "Customer Usage" is unused_*
- b. Wake pin cannot be "yes" if "Selected Pin Direction" is output or non assigned.

6. 3.3 Tolerance Enable Check

- a. REV_SEL cannot be "enable" if the "customer usage" is unused_*

Appendix F Study of Terminologies

A. System on a Chip (SoC)

A SoC is a microchip that integrates all the components of an electronic device into a single chip. It is known for low power consumption and is widely used in mobile phones and embedded systems.

B. CUDA

CUDA is a parallel computing platform and API model created by NVIDIA.

C. Microprocessor and Microcontroller

A Microprocessor only has the CPU inside the chip. It does not have ROM, RAM, or any other peripherals on the chip.

A Microcontroller contains all the peripherals on one small chip. It is a synonym with System on a Chip.

Appendix G Tegra default pinmux CFG File

```
##
## Pinmux version 1.0
## Input pinmux file name:
/home/smangipudi/shared/pinmux/16_mar_2016/tegra18x-p3310-1000-a0-a2-cvm-config-pinmux.dtsi
## Input gpio file name:
/home/smangipudi/shared/pinmux/16_mar_2016/tegra18x-p3310-1000-a0-a2-cvm-config-gpio-default.dtsi
## Generation date: 2016-03-16 13:50
## PLEASE DO NOT EDIT THIS FILE
## This is autogenerated file using the script pinmux-dts2cfg.py
##
pinmux.major = 1;
pinmux.minor = 0;
##### Pinmux for gpio-input pins #####
pinmux.0x022130a0 = 0x00000001; # CONFIG B5
pinmux.0x02434050 = 0x00000000; # GPIO gpio_wan2_pb5
pinmux.0x02213200 = 0x00000001; # CONFIG C0
pinmux.0x02434040 = 0x00000000; # GPIO gpio_wan4_pc0
pinmux.0x022150a0 = 0x00000001; # CONFIG J5
pinmux.0x02431018 = 0x00000000; # GPIO gpio_aud0_pj5
pinmux.0x022156a0 = 0x00000001; # CONFIG M5
pinmux.0x02432028 = 0x00000000; # GPIO dmic4_clk_pm5
pinmux.0x02210880 = 0x00000001; # CONFIG I4
pinmux.0x02433020 = 0x00000000; # GPIO gpio_pq4_pi4
pinmux.0x022108c0 = 0x00000001; # CONFIG I6
pinmux.0x02433030 = 0x00000000; # GPIO gpio_pq6_pi6
pinmux.0x022108e0 = 0x00000001; # CONFIG I7
pinmux.0x02433038 = 0x00000000; # GPIO gpio_pq7_pi7
pinmux.0x0c2f1020 = 0x00000001; # CONFIG FF1
pinmux.0x0c301000 = 0x00000000; # GPIO gpio_sw1_pff1
pinmux.0x0c2f1040 = 0x00000001; # CONFIG FF2
pinmux.0x0c301008 = 0x00000000; # GPIO gpio_sw2_pff2
pinmux.0x0c2f1060 = 0x00000001; # CONFIG FF3
pinmux.0x0c301010 = 0x00000000; # GPIO gpio_sw3_pff3
pinmux.0x0c2f1080 = 0x00000001; # CONFIG FF4
pinmux.0x0c301018 = 0x00000000; # GPIO gpio_sw4_pff4
pinmux.0x0c2f1000 = 0x00000001; # CONFIG FF0
pinmux.0x0c301058 = 0x00000000; # GPIO power_on_pff0
pinmux.0x022114e0 = 0x00000001; # CONFIG X7
pinmux.0x0243d058 = 0x00000000; # GPIO uart5_cts_px7
```

```

pinmux.0x02211600 = 0x00000001; # CONFIG Y0
##### Pinmux for used pins #####
pinmux.0x02434038 = 0x00000400; # uart4_tx_pb0: uartd, tristate-disable, input-disable
pinmux.0x02434030 = 0x00000458; # uart4_rx_pb1: uartd, pull-up, tristate-enable, input-enable
pinmux.0x02434028 = 0x00000400; # uart4_rts_pb2: uartd, tristate-disable, input-disable
pinmux.0x02434020 = 0x00000458; # uart4_cts_pb3: uartd, pull-up, tristate-enable, input-enable
pinmux.0x02434018 = 0x00000440; # dap2_sclk_pc1: i2s2, tristate-disable, input-enable
pinmux.0x02434008 = 0x00000440; # dap2_dout_pc2: i2s2, tristate-disable, input-enable
pinmux.0x02434000 = 0x00000450; # dap2_din_pc3: i2s2, tristate-enable, input-enable
pinmux.0x02434010 = 0x00000440; # dap2_fs_pc4: i2s2, tristate-disable, input-enable
pinmux.0x02431040 = 0x00000440; # dap1_sclk_pj0: i2s1, tristate-disable, input-enable
pinmux.0x02431038 = 0x00000440; # dap1_dout_pj1: i2s1, tristate-disable, input-enable
pinmux.0x02431030 = 0x00000450; # dap1_din_pj2: i2s1, tristate-enable, input-enable
pinmux.0x02431028 = 0x00000440; # dap1_fs_pj3: i2s1, tristate-disable, input-enable
pinmux.0x02431020 = 0x00000400; # aud_mclk_pj4: aud, tristate-disable, input-disable
pinmux.0x02431008 = 0x00000401; # gpio_aud2_pj7: dspk1, tristate-disable, input-disable
pinmux.0x02431000 = 0x00000401; # gpio_aud3_pk0: dspk1, tristate-disable, input-disable
pinmux.0x02432000 = 0x00004441; # dmic1_clk_pm1: i2s3, tristate-disable, input-enable
pinmux.0x02432008 = 0x00004451; # dmic1_dat_pm0: i2s3, tristate-enable, input-enable
pinmux.0x0243602c = 0x00000448; # sdmmc4_dat0: pull-up, tristate-disable, input-enable
pinmux.0x02436028 = 0x00002448; # sdmmc4_dat1: pull-up, tristate-disable, input-enable
pinmux.0x02436024 = 0x00002448; # sdmmc4_dat2: pull-up, tristate-disable, input-enable
pinmux.0x02436020 = 0x00002448; # sdmmc4_dat3: pull-up, tristate-disable, input-enable
pinmux.0x0243601c = 0x00000448; # sdmmc4_dat4: pull-up, tristate-disable, input-enable
pinmux.0x02436018 = 0x00002448; # sdmmc4_dat5: pull-up, tristate-disable, input-enable
pinmux.0x02436014 = 0x00002448; # sdmmc4_dat6: pull-up, tristate-disable, input-enable
pinmux.0x02436010 = 0x00002448; # sdmmc4_dat7: pull-up, tristate-disable, input-enable
pinmux.0x0243600c = 0x00000450; # sdmmc4_dqs: tristate-enable, input-enable
pinmux.0x0c301080 = 0x00000401; # gpio_dis0_pu0: gp, tristate-disable, input-disable
pinmux.0x0c301048 = 0x00000458; # batt_oc_ps2: soc, pull-up, tristate-enable, input-enable
pinmux.0x0243d020 = 0x00000400; # uart2_tx_px0: uartb, tristate-disable, input-disable
pinmux.0x0243d028 = 0x00000458; # uart2_rx_px1: uartb, pull-up, tristate-enable, input-enable
pinmux.0x0243d030 = 0x00000400; # uart2_rts_px2: uartb, tristate-disable, input-disable
pinmux.0x0243d038 = 0x00000458; # uart2_cts_px3: uartb, pull-up, tristate-enable, input-enable
pinmux.0x0243d048 = 0x00000452; # uart5_tx_px4: nv, tristate-enable, input-enable
pinmux.0x0243d078 = 0x00000401; # gpio_mdm4_py3: spi1, tristate-disable, input-disable
pinmux.0x0243d0b0 = 0x00000440; # gen7_i2c_scl_pl0: i2c7, tristate-disable, input-enable,
pinmux.0x0c302040 = 0x00000400; # uart7_tx_pw6: uartg, tristate-disable, input-disable
pinmux.0x0c302038 = 0x00000458; # uart7_rx_pw7: uartg, pull-up, tristate-enable, input-enable
pinmux.0x0c302050 = 0x00000400; # gpio_sen1_pv1: spi2, tristate-disable, input-disable
pinmux.0x0c302058 = 0x00000450; # gpio_sen2_pv2: spi2, tristate-enable, input-enable
pinmux.0x0c302060 = 0x00000400; # gpio_sen3_pv3: spi2, tristate-disable, input-disable
pinmux.0x0c302068 = 0x00000400; # gpio_sen4_pv4: spi2, tristate-disable, input-disable
pinmux.0x0c302078 = 0x00000401; # gpio_sen6_pv6: gp, tristate-disable, input-disable
pinmux.0x0c302080 = 0x00000401; # gpio_sen7_pv7: wdt, tristate-disable, input-disable

```

Appendix H Screenshots of GUI Tool

TEGRA PINMUX GUI TOOL | Import File | Export File | Generate CFG

Categories

- AO (19)
- AO_HV (3.3V Capable) (12)
- AUDIO (9)
- AUDIO_HV (3.3V Capable) (10)
- CAM (11)
- CONN (14)
- DEBUG (4)
- DMIC_HV (8)
- EDP (7)
- PEX_CTL (7)
- SDMMC1_HV (3.3V Capable) (5)
- SDMMC3_HV (3.3V Capable) (5)
- SPI (6)
- SYS (46)
- UFS (2)

Pinmux Dashboard

Ball Name	Category	Customer Usage	Pin Direction	Req. Initial State	Wake Pin	3.3V Tolerance E...	LPDR Enable	IO Block V...
CONN								
UART4_TX	CONN	UD3_TXD	Output				Disable	1.8V
UART4_RX	CONN	UD3_RXD	Input	Int PU			Disable	
UART4_RTS	CONN	UD3_RTS	Output				Disable	
UART4_CTS	CONN	UD3_CTS	Input	Int PU	No		Disable	
GPIO_WAN1	CONN	GPIO3_PB.04	Output	Drive 1			Disable	
GPIO_WAN2	CONN	GPIO3_PB.05	Input	Int PU	Yes		Disable	
GPIO_WAN3	CONN	GPIO3_PB.06	Output	Drive 0			Disable	
GPIO_WAN4	CONN	GPIO3_PC.00	Input	Int PU	Yes		Disable	
DAP2_SCLK	CONN	I2S2_SCLK	Bidirectional				Disable	
DAP2_DOUT	CONN	I2S2_SDATA_OUT	Bidirectional				Disable	
DAP2_DIN	CONN	I2S2_SDATA_IN	Input				Disable	
DAP2_FS	CONN	I2S2_LRCK	Bidirectional				Disable	
GEN1_I2C_SCL	CONN	I2C1_CLK	Bidirectional			Enable	Disable	
GEN1_I2C_SDA	CONN	I2C1_DAT	Bidirectional		No	Enable	Disable	
AUDIO								

Copyright (c) 2017, NVIDIA CORPORATION. All rights reserved | [Legal Information](#) | [Privacy Policy](#) | [Help Page](#)

Figure H (a) Pinmux Tab

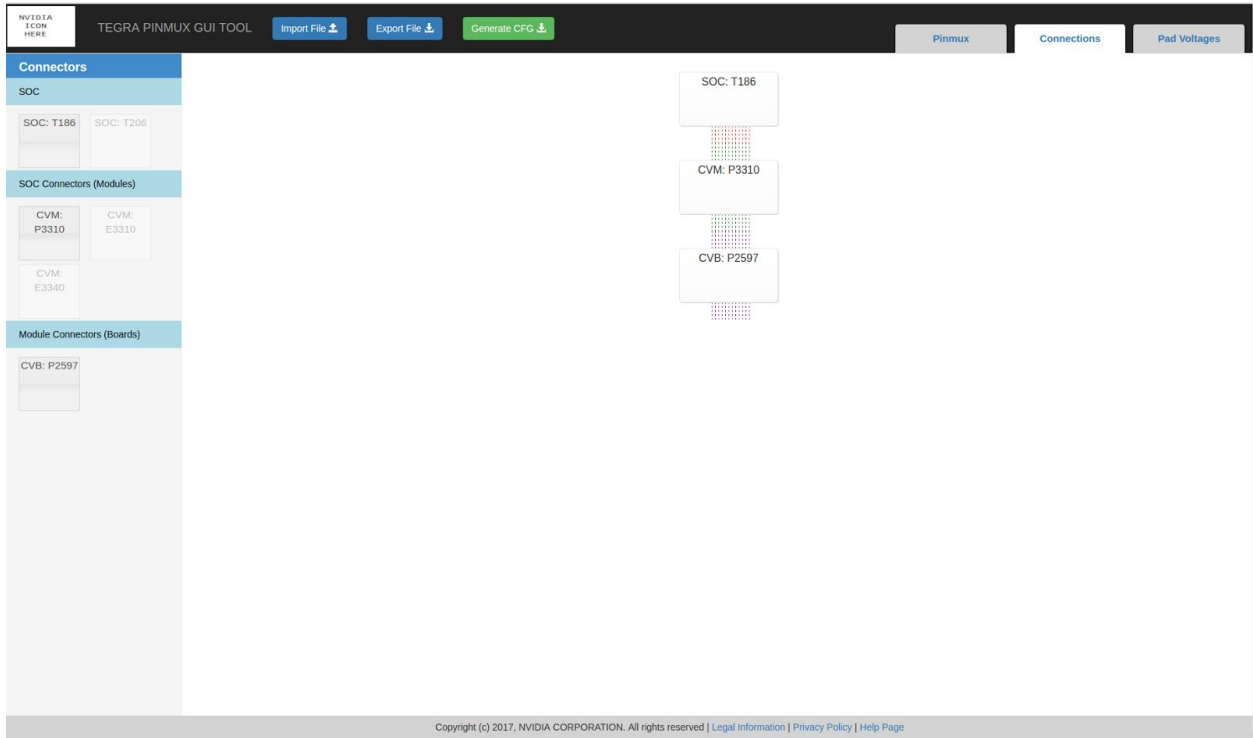


Figure H (b) Connections Tab

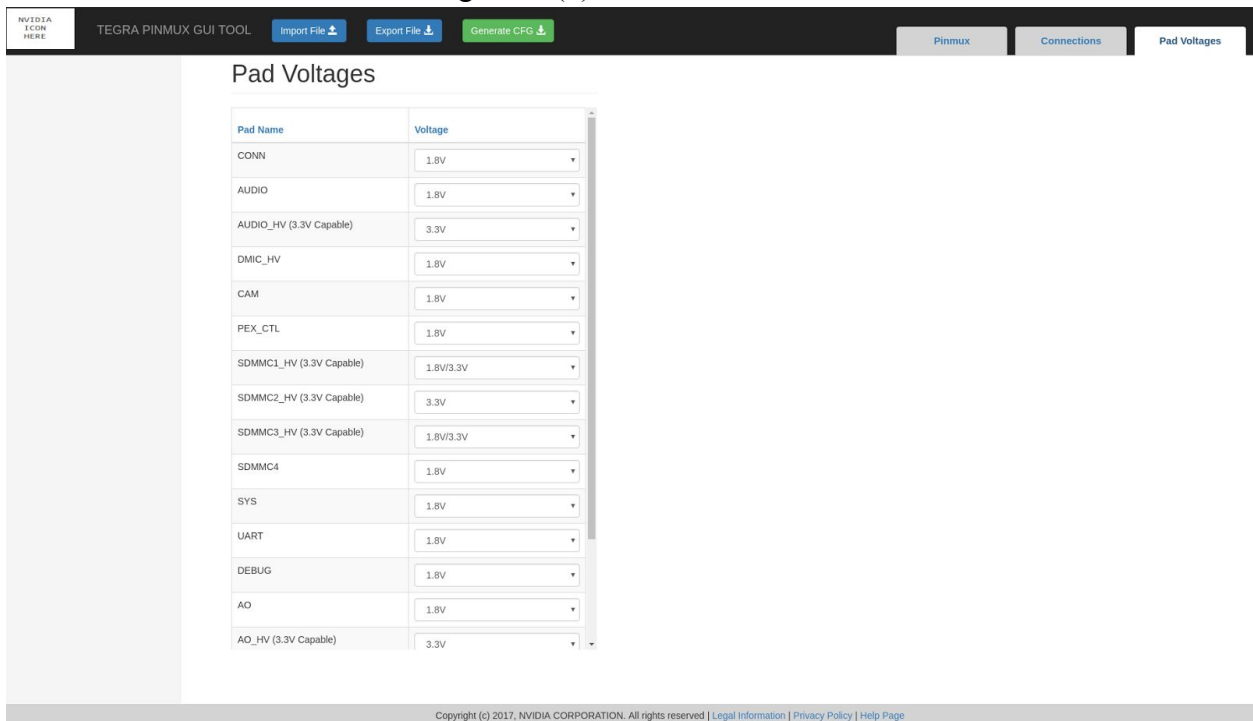


Figure H (c) Pad Voltages Tab

Pinmux GUI Tool Help Page

What is the Tegra Pinmux GUI Tool?

This is a local web app GUI tool which allows configuration of pinmux and pad voltages on the Jetson-TX2 development board.

The tool reads in data files for a Tegra board, and allows users to change the board connections, configure the pinmux settings as well as the pad voltages. The tool can output a zip file containing cfg files that can be flashed onto the MB1.

The tool will apply data validation rules to prevent users from configuring pins incorrectly, and allow constraints - for when a user wants to modify pins pertaining to certain board connections.

The tool allows users to save their board and pinmux configurations by downloading a txt configuration file. They can import that file into the tool later and restore their configurations on the webpage.

The tool seeks to replace the old method of editing spreadsheets on Windows, exporting DTS files, moving them to Linux, and running some Python scripts to convert them to CFG files. Instead it can generate CFG files on any platform.

What does the Pinmux page do?

The pinmux dashboard displays a table containing all the pinmux configurations for the TX2 board. The data is read from a data file listing all the constraints and data. The sidebar displays the pinmux categories and can filter certain categories to be visible in the table. The user can hover over a table column header to see a brief description of the column name. The table will highlight any kind of error if the user configures a pin incorrectly. Each error is highlighted with a red border around the table cell.

What does the Connections page do?

The connections page displays the SOC board as a starting point. The user can drag a connector from the sidebar to the board if it is allowed, which will change the constraints in the pinmux table. The user can also drag a connector from the board to the sidebar to remove it. The boards must be added and removed in the correct order.

What does the Pad Voltages page do?

The pad voltages page displays a table with a voltage for each category. The voltage is global and is set for all the pins in that category. Once the user changes a voltage, it will also be changed under the IO Block Voltage column in the pinmux page.

How do I change the IO Block Voltages in the pinmux page?

The pinmux table only displays the voltages in text. To edit the voltages, the user should go to the voltages tab.

Why can't I edit the pinmux table?

If a table cell is greyed out, it means this pin has been constrained by the device tree in the connections page. If the user

Copyright (c) 2017, NVIDIA CORPORATION. All rights reserved. | [Legal Information](#) | [Privacy Policy](#) | [Help Page](#)

Figure H (d) Help Page

Appendix I Instructions to generate JSON Data file from Excel

Spreadsheet

Instructions on How to Generate the Files in /pinmux/data

1. Download the spreadsheets -----

Link to the spreadsheets on p4 (only NVIDIA employees have access)

For soc.txt, download T186_IC_Generic_Customer_Pinmux_Release.xlsm.

For cvm.txt or cvb.txt, download Quill-CVM-Generic-Customer-Pinmux-Template.xlsm.

These spreadsheets are also available to the NVIDIA customers.

2. Modify the headers of the spreadsheets -----

Note: DO NOT edit any other content and keep the default pin configurations.

Step 1: Unhide all the columns in the spreadsheet.

Step 2: Under the "Pin Muxing" Section there are five column headers, GPIO and SFIO0 to SFIO3.

Add the prefix "Pinmux" to all of them, so they become PinmuxGPIO, PinmuxSFIO0, PinmuxSFIO1, PinmuxSFIO2, PinmuxSFIO3 and PinmuxSFIO4.

Step 3 (Only for Quill-CVM-...): Rename the cell A9 from "Signal Name" to "CONN".

Step 4 (Only for Quill-CVM-...): Remove the row 8, which only contains "MPIO" in column C.

3. Select and copy the spreadsheet -----

Step 1: For row selections, start from row 7 (which contains the headers Ball Name, DSC, MID...),
until the last row that defines pin configurations. Do not include any empty row after that.
Do not include the small section "Voltage Configuration" at the very bottom.

Step 2: For column selections, do not include the last column, which is called
"Recommended Usage Description" in T186_IC_Generic_... or "Single-cell Tablet usage
8 Lane
DSI Panel" In Quill-CVM-...

(TODO: Give instructions on how to generate the column "Apply Constraint")

Step 3: Copy the whole selection.

4. Paste the spreadsheet to the website and generate files

Step 1: Open the website <http://www.convertcsv.com/csv-to-json.htm>.

Step 2: Paste the selection into the text box under "Step 1: Select your input".

Step 3: Scroll to "Step 5: Generate output" and click the button "CSV To JSON".

Step 4: Click the button "Download Result", and the browser will download a JSON file.

Step 5: Rename the file to "soc.txt" (or any other depending on the spreadsheet used).

Step 6: Replace the file with the same name in pinmux/data/. Refresh the Pinmux Tool webpage.

Other Information

When this instruction file is written, the spreadsheet T186_IC_Generic_... used is Revision #1,
and the spreadsheet Quill-CVM-... used is Revision #6. If the spreadsheets are updated, some
instructions may need to change.

The file pads.txt was written by hand. It keeps a list of the pad names shown in the "Voltage
Configuration"
section, which can be found at the very bottom of any spreadsheet (all the columns should be
visible).

Since this section differs in each spreadsheet, pads.txt will keep most of the pad names that have shown up.

The Pinmux Tool Webpage will read from pads.txt to write to pad.dts.