NXP Challenge

October, 2025

Edgar Saul Valencia Olvera

Santiago de Querétaro, Querétaro. México.

Index

Abstract

This report shows the main concepts that I considered essential to understand the development of the NXP Challenge. It also includes a list of the main commands and their purposes, which were important to build the base of the project. In addition, the report explains how to create the Git repository and make the first commit, starting the version control process of the development.

1. Introduction

This NXP challenge asks you to create an embedded system that simulates a temperature sensor in the Linux kernel and makes it available to user applications. Before we look at the challenge details, let's understand the NXP i.MX processors that are commonly used in advanced embedded systems.

NXP i.MX Processors

NXP's i.MX processors are a family of application processors based on ARM technology. They combine multimedia capabilities, advanced connections, and low power consumption [1]. These processors are designed for:

- Edge computing and IoT devices [2].
- Human-machine interfaces (HMI) [2].
- Industrial and automotive applications [2].
- Vision systems and smart devices [2].

NXP provides complete software packages (Board Support Packages) for Linux that include kernels, drivers, and tools tested for these processors. This helps developers save time when building their applications [3].

For example:

The i.MX 8 series is made for industrial and vision systems with support for multiple displays and machine learning capabilities [4]

The newer i.MX 9 family includes modern cores, built-in AI acceleration, and enhanced security features [5].

About the main propose of the challenge, it is to create a kernel module called ***nxp_simtemp*** that simulates a temperature sensor with customizable behavior. This virtual sensor should be accessible from user applications through a special device file.

The challenge needs to:

- Generate temperature readings at regular intervals
- Provide data through a binary structure containing timestamp, temperature value, and status flags
- Notify applications when new data is available or when temperature crosses limits

- Allow configuration of settings like sampling rate and temperature thresholds
- Include a user application (CLI in Python or C++) that reads and displays the data
- Provide build and demo scripts for easy testing
- Include clear documentation about the design and implementation

2. Objective

The goal of this challenge is to demonstrate my skills in developing Linux drivers for embedded devices like the NXP i.MX processor. By creating a simulated temperature sensor driver, I hope to better understand how hardware and software work together in a Linux environment. This project also allows me to apply concepts like kernel-level programming, device driver, and communication between user space and the kernel.

Although I have not had the opportunity to program the Linux kernel directly or develop a CLI application, my goal is to learn the basic tools and fundamentals to make a good progress as possible in this challenge and get new experience throughout the process.

3. Methodology

The NXP Challenge methodology begins with environment setup by verifying the Ubuntu version and installing the necessary development tools including compilers, kernel headers, Git, and Python. A structured project directory is created and initialized with Git version control, then connected to a GitHub repository. The core development focuses on building a kernel driver incrementally in six steps: starting with a basic skeleton that loads and unloads, creating a character device at for data access, implementing a ring buffer with a timer to generate periodic temperature samples, adding poll/select support for efficient non-blocking reads, creating a sysfs interface for runtime configuration and adding Device Tree support with threshold detection for alerts. A Python CLI is developed to read and display temperature data. Shell scripts automate the build and testing process, while the documentation is written. A two-to-three-minute video demonstration is recorded showing compilation. In the following figure, a diagram of the NXP challenge methodology is presented.

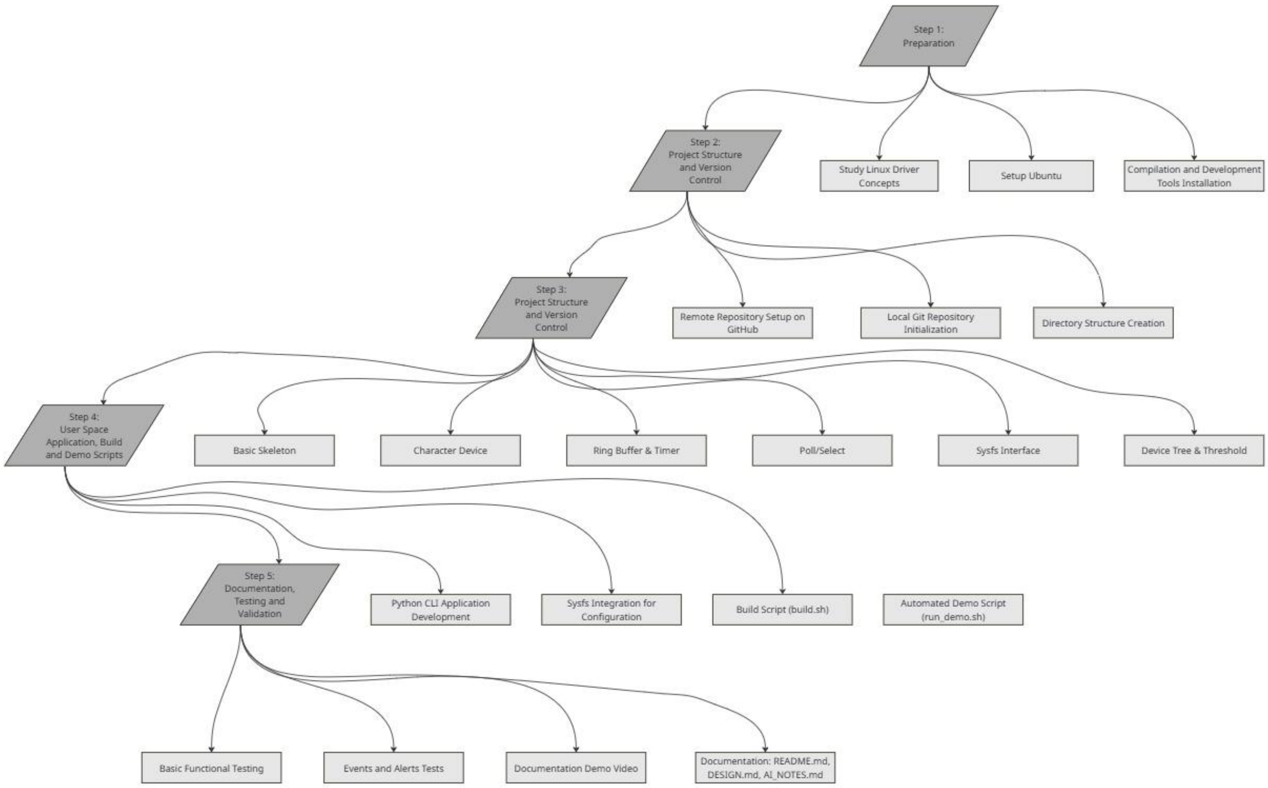

Figure. - 1 NXP Challenge Methodology.

4. Development

4.1 Concepts

1 **Linux Kernel**

The Linux kernel is the core component of the Linux operating system that acts as an intermediary between computer hardware and software applications. It manages system resources, provides essential services, handles process scheduling, memory management, device control, and networking functionalities. The kernel operates in a privileged execution mode with direct access to hardware, while enforcing security boundaries between processes and the system itself [6].

2 Device Drivers

Device drivers are specialized kernel modules that enable the operating system to communicate with hardware devices. They translate generic operating system commands into device-specific operations, providing abstraction that allows applications to interact with diverse hardware through standardized interfaces. Drivers handle device initialization, data transfer, interrupt servicing, and power management, essentially serving as translators between the kernel's generic I/O subsystem and specific hardware controllers [7].

3 Device Tree

The Device Tree is a data structure and language for describing hardware configuration in embedded systems. It replaces hard-coded device information in the kernel source code by providing a dynamic, portable mechanism to describe system hardware layout. The Device Tree source files (.dts) are compiled into binary blobs (.dtb) that the kernel parses during boot to discover and initialize hardware components without requiring code modifications for different board variants [8].

4 Kernel Modules

Kernel modules are dynamically loadable object files that extend kernel functionality without requiring system reboot or kernel recompilation. They enable the addition of device drivers, filesystems, and other kernel features at runtime using insmod or rmmod commands. Modules operate with full kernel privileges but are isolated from the core kernel, allowing for modular development and reducing the memory footprint by loading only required functionality [9].

4.2 Setup ubuntu and development tools installation

Installation of everything needed for kernel module development. In the following tables, the commands used to update the system and install the modules needed for kernel module development are shown.

Table. - 1 Table of commands to update and enable the Linux kernel

Command	Purpose
`sudo apt update`	Updates the list of available packages from Ubuntu repositories
`sudo apt upgrade -y`	Upgrades all installed packages to their latest versions
`sudo apt install -y build-essential`	Installs essential compilation tools including gcc, g++, make, and basic libraries
`sudo apt install -y linux-headers-$(uname -r)`	Installs kernel headers for the current kernel version (required to compile kernel modules)
`sudo apt install -y kmod`	Installs module management tools (insmod, rmmod, lsmod, modinfo)
`sudo apt install -y linux-tools-common`	Installs additional common kernel tools
`sudo apt install -y linux-tools-generic`	Installs generic Linux tools for kernel development
`sudo apt install -y dkms`	Installs Dynamic Kernel Module Support for automatic module rebuilding
`sudo apt install -y bc`	Installs basic calculator tool needed for kernel compilation
`sudo apt install -y bison`	Installs parser generator tool required for building the kernel
`sudo apt install -y flex`	Installs lexical analyzer tool needed for kernel compilation
`sudo apt install -y libelf-dev`	Installs library for handling ELF executable files
`sudo apt install -y libssl-dev`	Installs SSL development library for cryptography support

`sudo apt install -y libncurses-dev`	Installs library for creating text-based user interfaces
`sudo apt install -y cppcheck`	Installs static code analysis tool for C/C++
`sudo apt install -y clang-format`	Installs code formatting tool for consistent code style
`sudo apt install -y sparse`	Installs semantic checker for C code
`mkdir -p ~/kernel-scripts`	Creates a directory to store kernel verification scripts
`wget -O ~/kernel-scripts/checkpatch.pl [URL]`	Downloads the kernel code style checker script
`wget -O ~/kernel-scripts/spelling.txt [URL]`	Downloads spelling dictionary for checkpatch
`wget -O ~/kernel-scripts/const_structs.checkpatch [URL]`	Downloads constant structures list for checkpatch
`chmod +x ~/kernel-scripts/checkpatch.pl`	Makes the checkpatch script executable
`echo 'export PATH=$PATH:~/kernel-scripts' >> ~/.bashrc`	Adds kernel-scripts directory to system PATH permanently
`source ~/.bashrc`	Reloads bash configuration to apply PATH changes immediately
`gcc --version`	Checks if GCC compiler is installed and shows its version
`make --version`	Checks if Make build tool is installed and shows its version
`ls -la /lib/modules/$(uname -r)/build`	Verifies that kernel build directory exists
`ls -la /lib/modules/$(uname -r)/build/include`	Verifies that kernel header files are present
`which insmod`	Checks if insmod command is available (loads kernel modules)

`which rmmod`	Checks if rmmod command is available (removes kernel modules)
`which lsmod`	Checks if lsmod command is available (lists loaded modules)
`which modinfo`	Checks if modinfo command is available (shows module information)

4.3 GitHub repository and project structure creation

The following commands in the next table let us to install Git, check the installation, set up the information, and verify it.

Table. - 2 GitHub Commands

Command	Purpose
`sudo apt update`	Updates the list of available packages and their versions
`sudo apt install -y git`	Installs Git on your system automatically (the -y flag answers yes to all prompts)
`git --version`	Shows the Git version installed on your computer
`git config --global user.name "Your Name"`	Sets your name for all Git projects on your computer
`git config --global user.email "your_email@example.com"`	Sets your email address for all Git projects on your computer
`git config --list`	Shows all your current Git settings and configuration

In GitHub, we create our account and set up access with a Token. It is known that since Ubuntu 2021, GitHub requires Tokens instead of passwords. The repository is created with the name **NXP Simtemp Challenge**. With the following commands from the table below, the project structure is created.

Table. - 3 Project Structure

Command	Purpose
`mkdir -p ~/nxp-simtemp-challenge`	Creates a new folder in your home directory (the -p flag creates parent folders if needed)
`cd ~/nxp-simtemp-challenge`	Changes your current location to the project folder
`git init`	Starts a new Git repository in the current folder
`mkdir -p kernel/dts`	Creates a folder named "dts" inside the "kernel" folder
`mkdir -p user/cli`	Creates a folder named "cli" inside the "user" folder
`mkdir -p user/gui`	Creates a folder named "gui" inside the "user" folder
`mkdir -p scripts`	Creates a folder named "scripts"
`mkdir -p docs`	Creates a folder named "docs" (short for documents)
`touch kernel/Kbuild`	Creates an empty file named "Kbuild" in the kernel folder
`touch kernel/Makefile`	Creates an empty file named "Makefile" in the kernel folder
`touch kernel/nxp_simtemp.c`	Creates an empty C source code file
`touch kernel/nxp_simtemp.h`	Creates an empty C header file
`touch kernel/nxp_simtemp_ioctl.h`	Creates an empty header file for I/O control definitions
`touch kernel/dts/nxp-simtemp.dtsi`	Creates an empty device tree include file
`touch user/cli/main.py`	Creates an empty Python file for the command line interface
`touch user/cli/requirements.txt`	Creates an empty file to list Python dependencies
`touch scripts/build.sh`	Creates an empty shell script for building the project

`touch scripts/run_demo.sh`	Creates an empty shell script to run demonstrations
`touch scripts/lint.sh`	Creates an empty shell script for code checking
`touch docs/README.md`	Creates an empty markdown file for project documentation
`touch docs/DESIGN.md`	Creates an empty markdown file for design documentation
`touch docs/TESTPLAN.md`	Creates an empty markdown file for testing plans
`touch docs/AI_NOTES.md`	Creates an empty markdown file for AI-related notes
`touch .gitignore`	Creates an empty file that tells Git which files to ignore

Now the (**. gitignore)** file is created to avoid uploading files that are not necessary to share to the repository. The code is shown below.

```
nano .gitignore
----------------------------------------------------------------------
-----------------------------------------------
# Kernel build artifacts
*.o
*.ko
*.mod
*.mod.c
*.cmd
.tmp_versions/
modules.order
Module.symvers
*.o.d

# Python
```

```
__pycache__/
*.py[cod]
*$py.class
*.so
.Python
env/
venv/
.venv

# IDE
.vscode/
.idea/
*.swp
*.swo
*~

# Build directories
build/
dist/

# Logs
*.log
```

Now, with the following commands, we connect the local repository with GitHub, make the first commit, and check the created structure.

```
# Add to remote (ULR)
git remote add origin https://github.com/EdgarValencia95/nxp-simtemp-challenge.git

# Check the remote
git remote -v
```

```
# Add all files
git add .

# Now make the commit
git commit -m "Initial project structure for NXP simtemp challenge"

# Create the main branch and push
git branch -M main
git push -u origin main
```

Now, with the command **`tree -L 3`** we should see the project structure, and in the GitHub repository the same structure appears as shown in the following screenshot images.

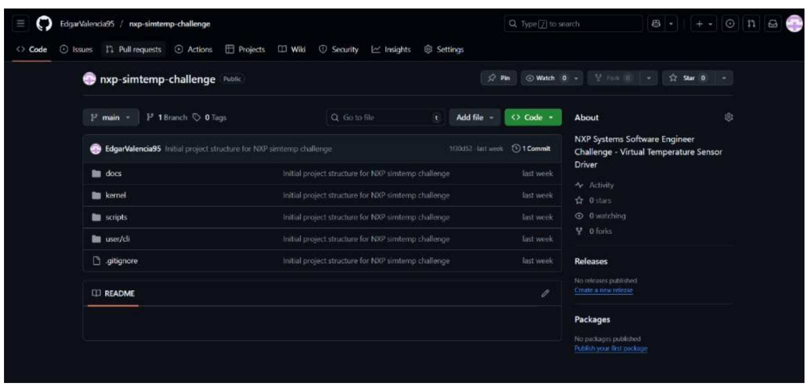

Figure. - 2 GitHub Repository

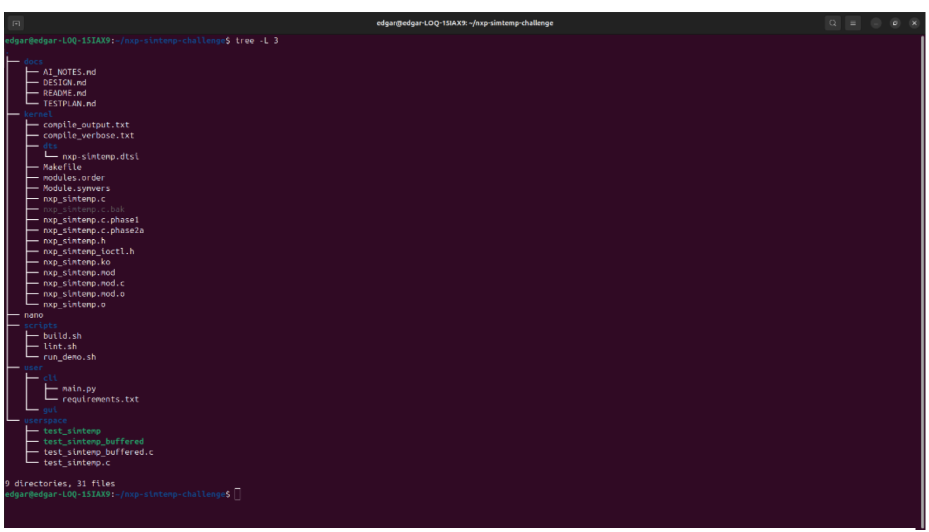

Figure. - 3 Local Repository

5. Conclusion

Personally, this challenge was a real test for me because I had never had the opportunity to work with kernel programming before. I had some experience programming processors like the Raspberry Pi and I had taken a few courses about the “Lagarto” microprocessor developed at the Instituto Politécnico Nacional (IPN). However, this project allowed me to explore a completely new programming environment. In my current work, I usually program microcontrollers, so my first challenge was to understand what a kernel is and look for information that could help me to understand what needed to be done. From there, I created a small methodology to organize my work and follow a sequence throughout the project.I know I didn’t complete all the points required by the challenge, but I tried to move forward step by step and understand the development process better. One of the best parts of this experience was learning how to use online repositories with examples and information on many topics, and how AI tools can make self-learning to much easier. Overall, this challenge was a great opportunity to learn, grow, and get new knowledge in an area that was completely new to me.

6. References

[1] N. Semiconductors, "i.MX Applications Processors — Arm Cortex-A based," NXP Semiconductors, [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors:IMX_HOME. [Accessed 10 October 2025].

[2] N. Semiconductors, "i.MX 8 Series Applications Processors Overview," NXP Semiconductors, [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-8-applications-processors:IMX8-SERIES. [Accessed 10 October 2025].

[3] N. Semiconductors, "Embedded Linux for i.MX Applications Processors," NXP Semiconductors, [Online]. Available: https://www.nxp.com/design/design-center/software/embedded-software/i-mx-software/embedded-linux-for-i-mx-applications-processors%3AIMXLINUX. [Accessed 10 October 2025].

[4] N. Semiconductors, "i.MX 8 Series Applications Processors Multicore Arm Cortex Processors," NXP Semiconductors, [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-8-applications-processors%3AIMX8-SERIES. [Accessed 10 October 2025].

[5] N. Semiconductors, "i.MX 93 Applications Processor Family – Arm® Cortex®-A55, ML Acceleration, Power Efficient MPU," NXP Semiconductors, [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-9-processors/i-mx-93-applications-processor-family-arm-cortex-a55-ml-acceleration-power-efficient-mpu%3Ai.MX93. [Accessed 10 October 2025].

[6] T. k. d. community, "The Linux Kernel Documentation," 2024. [Online]. Available: https://www.kernel.org/doc/html/latest/process/1.Intro.html. [Accessed 15 October 2025].

[7] A. R. a. G. K.-H. Jonathan Corbet, "Linux Device Drivers, Third Edition," 2005. [Online]. Available: https://lwn.net/Kernel/LDD3/. [Accessed 15 October 2025].

[8] D. Specification, "Document master project for the Devicetree Specification," Devicetree Specification, [Online]. Available: https://www.devicetree.org/. [Accessed 15 October 2025].

[9] R. Russell, "Unreliable Guide To Hacking The Linux Kernel," The kernel development community, [Online]. Available: https://www.kernel.org/doc/html/latest/kernel-hacking/hacking.html#unreliable-guide-to-hacking-the-linux-kernel. [Accessed 15 October 2025].