University of Colorado Boulder

December 5th 2025

ECEN 5803 Mastering Embedded System Architecture

Project 2

"VoIP Gateway Design Evaluation"

Authors:

Nimish Sabnis
Nimisha Madapura Rajashekhar
Hannah Caldwell Meurer

Under the Guidance of:

Prof. Timothy Scherr

Table Of Contents

Executive Summary

The VoIP Gateway Design project progressed through several structured phases to evaluate the Broadcom BCM2711 processor as a platform for VoIP gateway development. Through a series of structured tests across QNX, Linux, the team examined communication reliability, system behavior, telecom functionality, and IoT reporting capabilities.

The initial phase established reliable communication between the QNX operating system and the Momentics IDE, ensuring that the development environment and target interfaces were correctly configured. Connectivity was then validated through both serial and Ethernet access to the QNX target. With stable communication confirmed, the team successfully built and executed a C program including a basic decoder verifying the end-to-end build, deployment, and run process. During this stage, the team also identified key architectural distinctions between QNX and Linux, particularly QNX's microkernel, message-passing design, and real-time characteristics compared to Linux's monolithic and general-purpose approach.

Subsequent testing validated Linux as a client platform by demonstrating VoIP functionality using a softphone application. This confirmed proper call handling and communication paths in a practical usage scenario. Additional efforts expanded the system's capabilities by implementing IoT-style diagnostic services on QNX to report uptime, memory usage, and log information. These features establish a foundation for remote monitoring and management through a Python script, allowing the team to further analyze system behavior across QNX and Linux as well as compare implementations in Python and bash.

Overall, the BCM2711 processor demonstrated strong performance and compatibility across QNX, and Linux. Its stability, telecom-protocol support, and capacity for lightweight IoT reporting position it as a highly suitable for VoIP gateway deployments.

Problem Statement and Objectives

Patton is developing the model e911 IP-PBX VoIP Gateway, which needs an embedded systems platform capable of accurately measuring or handling IP networking, real time voice processing, and telephony signals. The company needs to determine whether the Broadcom BCM2711 chip is suitable for the product in terms of performance, cost, power consumption and I/O capabilities. The problem is to rigorously evaluate the processor, operating system and supporting software as well as to provide a system level diagram for the product.

Approach and Methodology for Evaluation

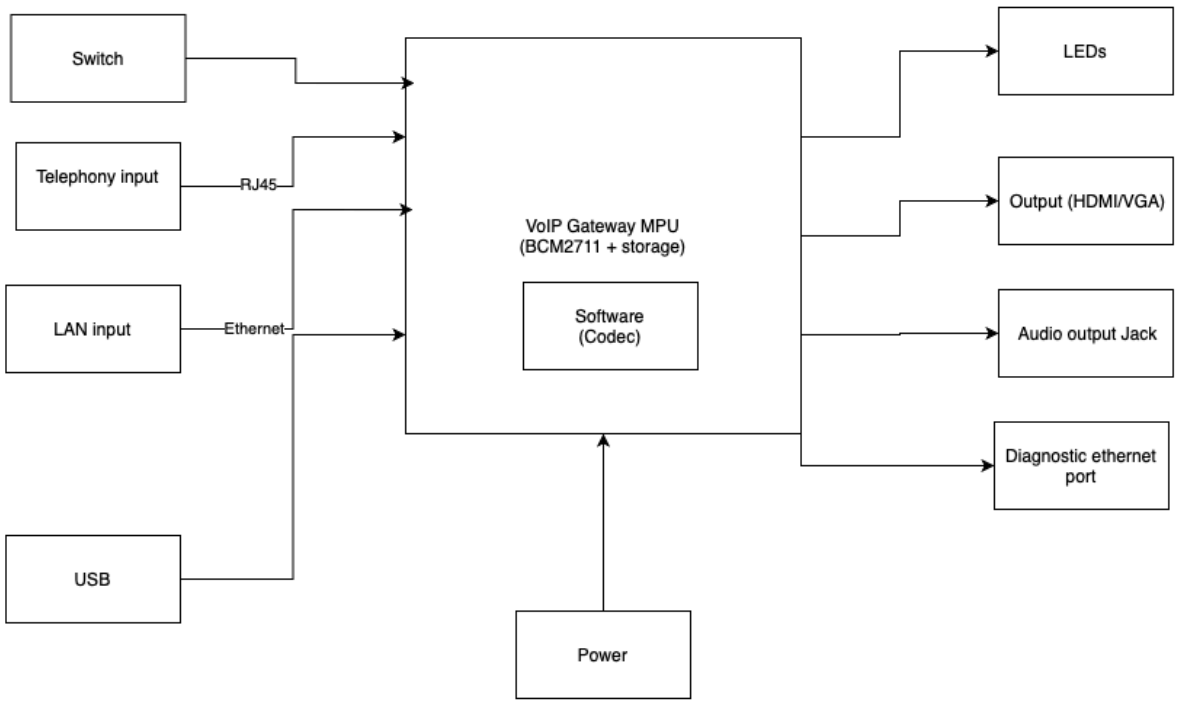

Figure 1: Block Diagram of VoIP Gateway

The work began with establishing a complete development environment for QNX, using QNX Software Center and Momentics IDE to configure the toolchain. Installation steps were verified and the installed components were logged to ensure reproducibility of the environment. This foundation allowed the team to proceed with systematic platform testing under QNX and Linux Operating systems.

The second stage involved preparing and booting the QNX image. The team then validated the system access through login, SSH, and the serial debug interface. Diagnostic commands were used to capture memory footprint, processor information and system temperature. A G.711 codec was then implemented and compiled in Momentics, and decoding tests were performed using reference audio files.

The third stage involved developing a linux based scripting module to quantify system resources under linux. A multi-functional monitoring script was written using Bash, incorporating diagnostic capabilities such as process inspection, CPU and memory summaries,network interface status and disk utilization. The script was executed both interactively and at boot time through crontab, allowing the team to analyze differences in system load, temperature and other resource consumption capabilities. These measurements provided an initial understanding of how system demand changes between idle, boot and user driven states.

In the fourth stage, an open-source PBX system was integrated into the linux environment to evaluate compatibility with VoIP requirements. Asterisk PBX system was installed, configured, and connected to a softphone application called Zoiper5 to test SIP registration, call handling and voicemail functionality. Screenshots of the registration logs, voicemail activation and call traces were collected to support validation.

In addition to this, a python based monitoring script was developed for QNX to directly compare system behaviour for the real time operating system. This provided a controlled

method for contrasting resource management and scheduling characteristics between QNX and Linux. The team also developed an IoT telemetry server on QNX for remote monitoring.

Module Test Results

In module 1, the QNX Software development platform (SDP) and momentics IDE were successfully installed and configured for ARM target. The installation process included getting QNX licence, setting up the SDP, adding required toolchains. Momentics was verified to detect appropriate compiler and project templates. A full list of installed components, along with screenshots of IDE configuration is provided in appendix

In module 2, the QNX image was booted up on the evaluation board successfully and the ethernet as well as the serial interface interaction was confirmed. The required diagnostic tests like capturing boot- time serial messages, verifying ethernet activity, and recording the memory footprint of the running system. The size of QNX found was 11Mb (ifs) + 2.4 Gb(System). A g.711 codec was written, compiled using Momentics and tested with an audio file provided for validation. The decoder produced correct output values. All measurements, terminal logs, and decoder output screenshots are included in appendix

In module 3, a multi-function script was developed and made executable to evaluate system behaviour under linux. The script has seventeen diagnostic functions covering process activity, CPU and memory utilization, thermal reading and health checks. Testing showed clear differences between terminal execution and boot time execution. It was observed that memory usage increased from 6% at boot to 26% in terminal, and CPU idle percentage rose from 47.8% at boot to 91.7% during terminal use. CPU temperature readings also demonstrated variation. The script was configured to run at startup using crontab, and all terminal outputs, boot logs, and quantitative comparisons are included in the appendix.

In module 4, the asterisk pbx was installed, configured and tested on linux OS to evaluate telephony capabilities relevant to the product. The installation was completed with standard build sequence, and asterisk was enabled as a system service. Memory sizing commands showed measurable storage usage across the installation directories, confirming that PBX stack used a significant amount of system resources. Voicemail for extension 100 was configured by modifying extensions.conf,pjsip.conf, and voicemail.conf, with SIP user 7001 registered through Zoiper using UDP transport on port 5060. A test call to extension 100 successfully reached the voicemail greeting, validating both SIP registration and dialplan operation. All changes to files, memory measurements and screenshots of Zoiper client and Asterisk CLI are included in the appendix. A video of this call is also provided with the deliverables.

In module 5, a QNX based IoT telemetry server was implemented and tested to evaluate remote monitoring capabilities for the product. The custom HTTP service exposed endpoints for CPU load, memory status, system uptime and an access log all retrieved using curl from local and remote clients. All endpoint outputs, logs and screenshots are provided in appendix

In module 7, a python script was developed to monitor key QNX system characteristics using built in diagnostic tools. The program provides an interactive menu that triggers different commands. It was found that python is a better scripting language for QNX. All captured outputs are provided in the appendix.

List of Project Deliverables

The submitted zip file consists of essential project deliverables like:

1. Comprehensive report which includes:
    1.1. Executive summary which includes detailed technical findings and performance measurements
    1.2. Problem Statement and Objectives
    1.3. Approach and Methodology with Block Diagram
    1.4. Module Test Results
    1.5. Recommendation of processor
2. Appendix which provides the screenshots , diagrams , videos and calculations of the questions in each Module.
3. Bill of Materials (BOM).
4. Software Design Files as Zip Folders containing the code.

Recommendations

For the VoIP Gateway Design Evaluation project, the Broadcom BCM2711 processor, as found in the Raspberry Pi 4, is recommended as the primary computing platform. While this MPU is highly capable, it is primarily available as a commercial product, and direct cost data for individual units was not obtainable. Therefore, for practical budgetary and procurement purposes, the BCM49408 is recommended as the alternative.

The BCM2711 remains a strong option due to its compatibility with both QNX and Linux, which enables flexibility across multiple operating environments. Its large community ecosystem and support for a wide range of open-source tools further streamline development and troubleshooting. However, it is most suitable for projects where sourcing from Raspberry Pi is feasible, typically larger-scale deployments.

The BCM49408, by contrast, meets budget constraints while providing sufficient processing power and memory resources. Its bill of materials totals $37.70. Key specifications include a low-power quad-core Cortex-A53 64-bit ARM processor running at 1.8 GHz, 1 MB shared L2 cache with ECC, 32-bit DDR3-1600 memory interface, 3× PCIe Gen2 lanes, USB3, SATA3, 2.5G PHY support (SGMII+), and 5 integrated Gigabit Ethernet PHYs with switch capabilities. Combined with Linux and QNX support, the BCM49408 represents a cost-effective and technically sufficient solution for achieving the objectives of the VoIP Gateway Design Evaluation project.

Appendix Project Module Questions

Appendix: Module 1 - Install QNX SDP and Momentics IDE

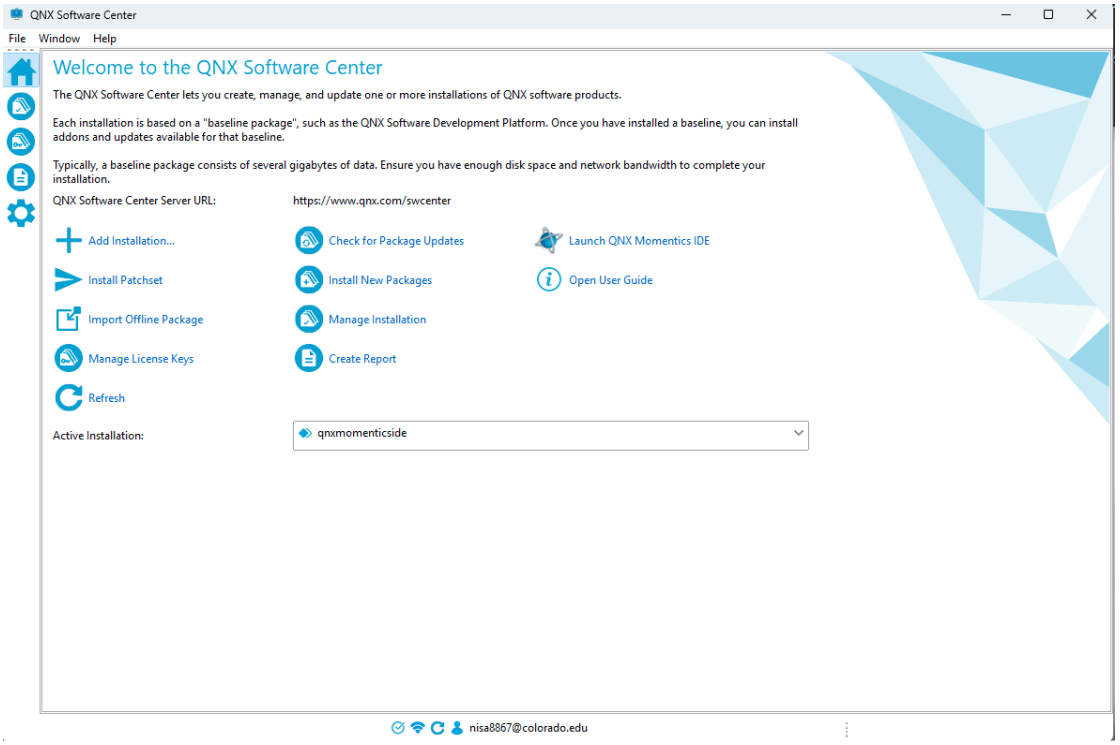

Figure 2: QNX Software centre

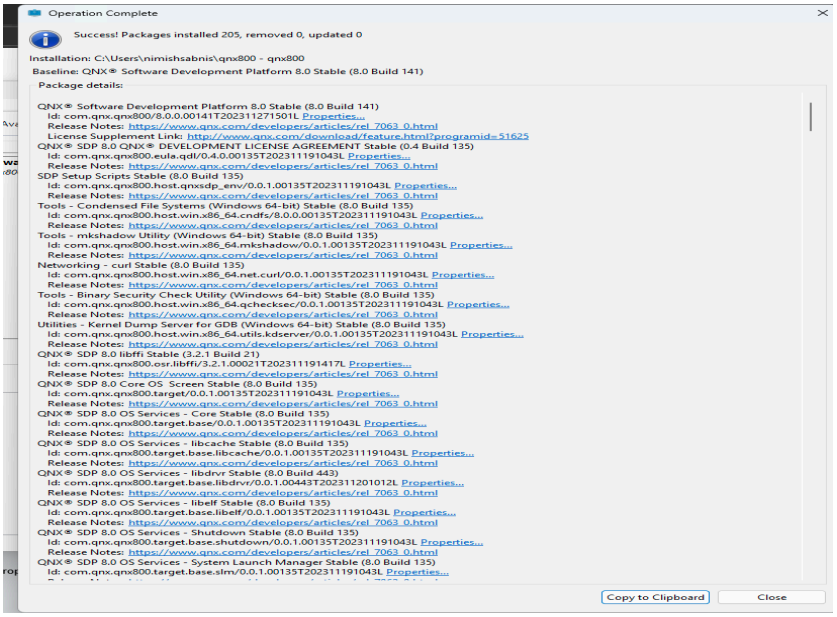

Figure 3: SDP 8.0

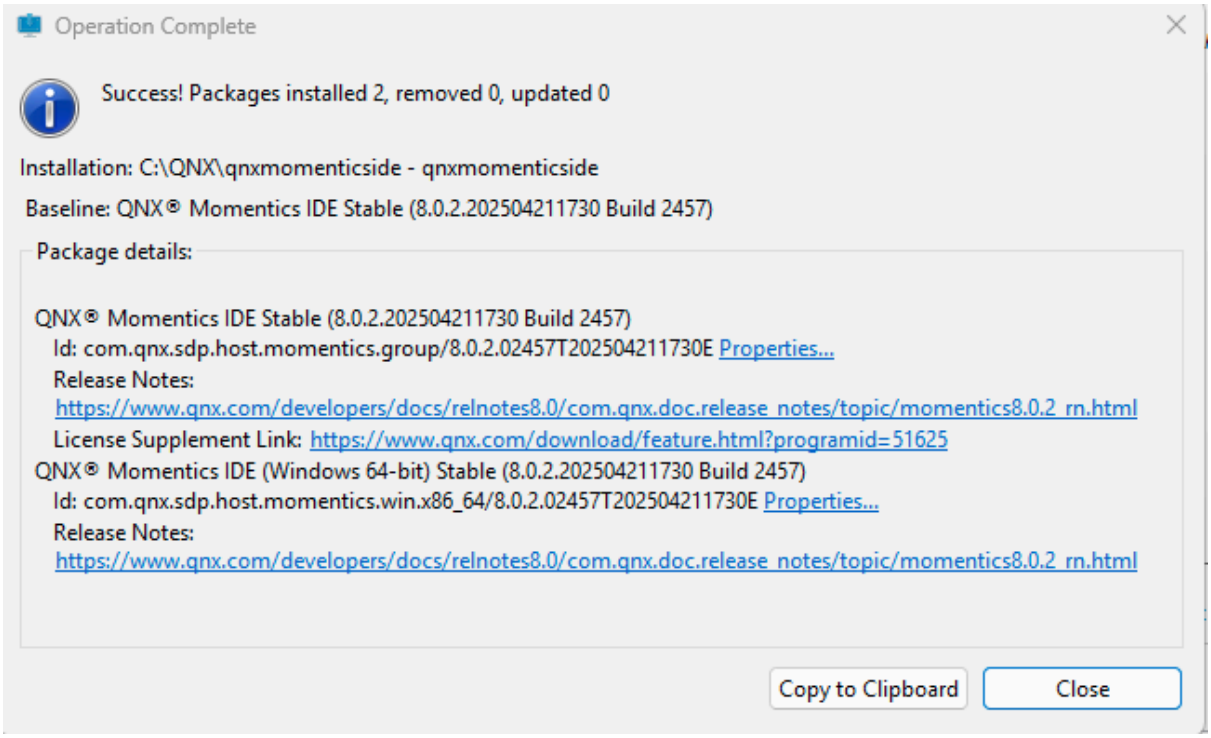

Figure 4: Momentics IDE installation

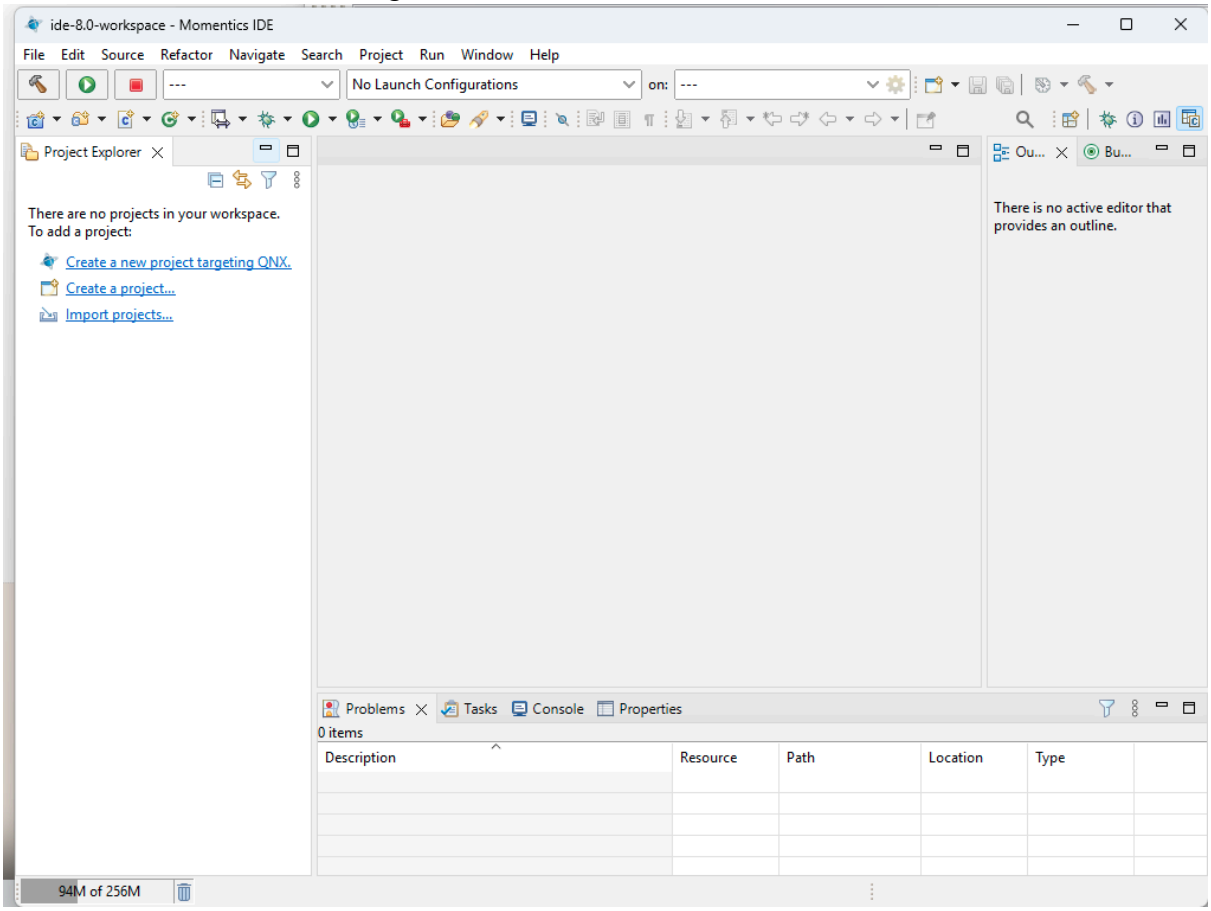

Figure 5: Momentics IDE

Appendix: Module 2 - BOOT QNX, Create G.711 CODEC

1. Booting RPi with QNX

The Images below show what the QNX OS looks like upon boot and after a successful login.

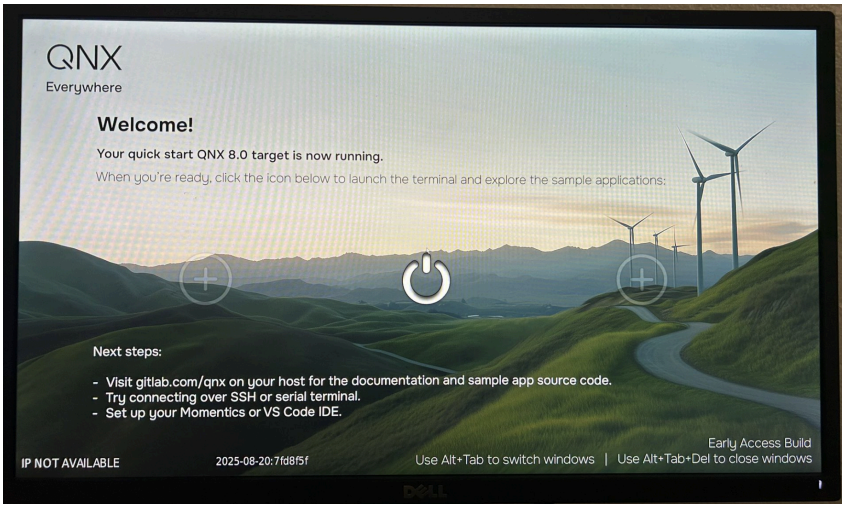

Figure 6: QNX Welcome Screen

Figure 7: QNX User Sign In

2. Serial Cable communication to serial tools

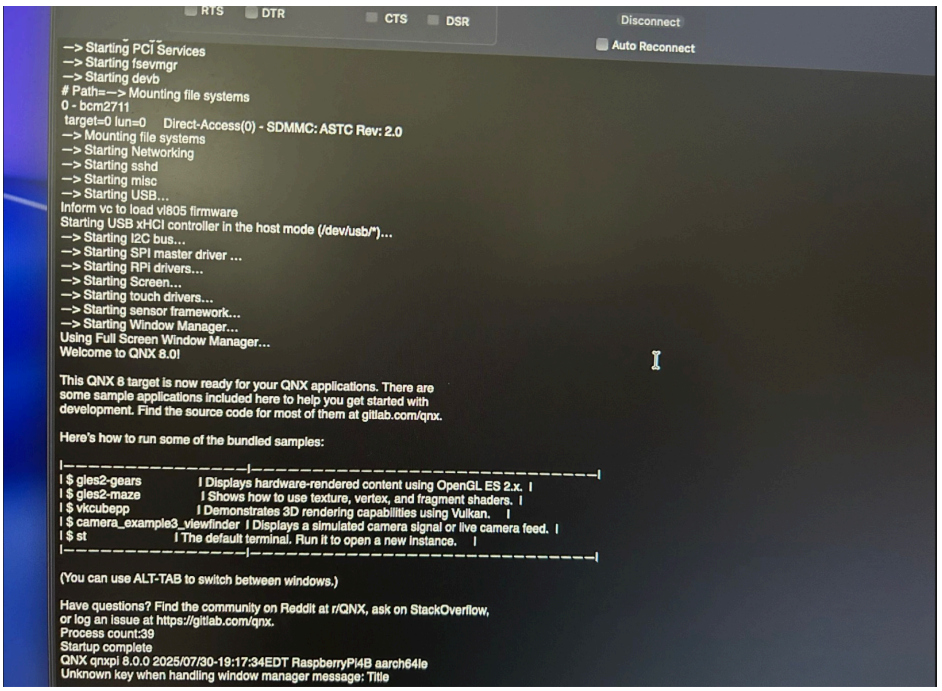

Figure 8: QNX Screen with Putty Connection via Serial Connection

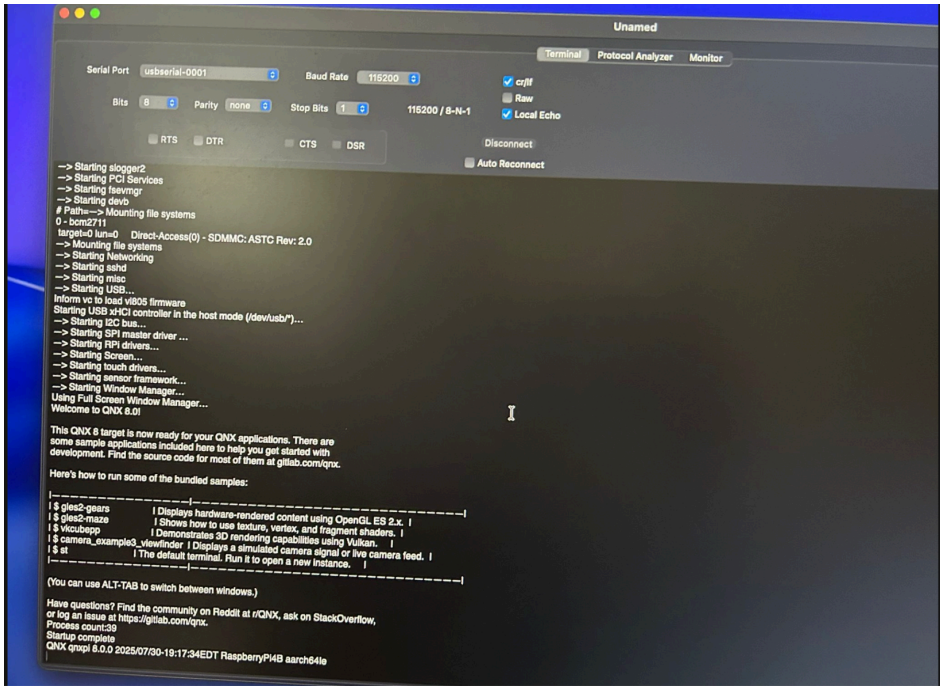

Figure 9: QNX Screen with Serial Cable Screen Capture #2

3. How much memory is used by the code? (What is the image size?)

The size of the image is 11Mb (ifs) + 2.4 Gb(System)

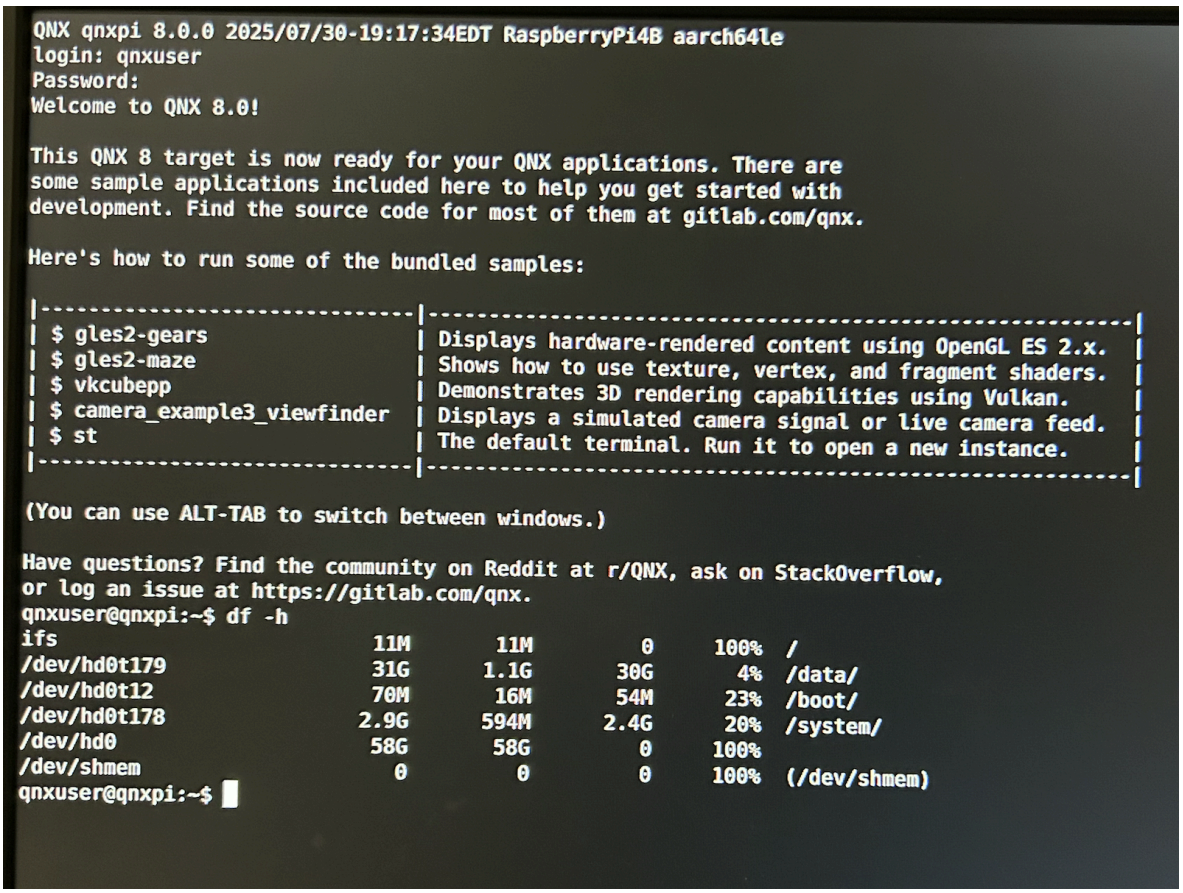

Figure 10: Screen Capture of Memory Usage

4. Ethernet communication

If one is using QNX while there is an Ethernet connection, the IP address can be seen in the bottom left-hand corner of the welcome screen. Then, on a personal computer using either Putty or a similar application, the UI for QNX can be seen.

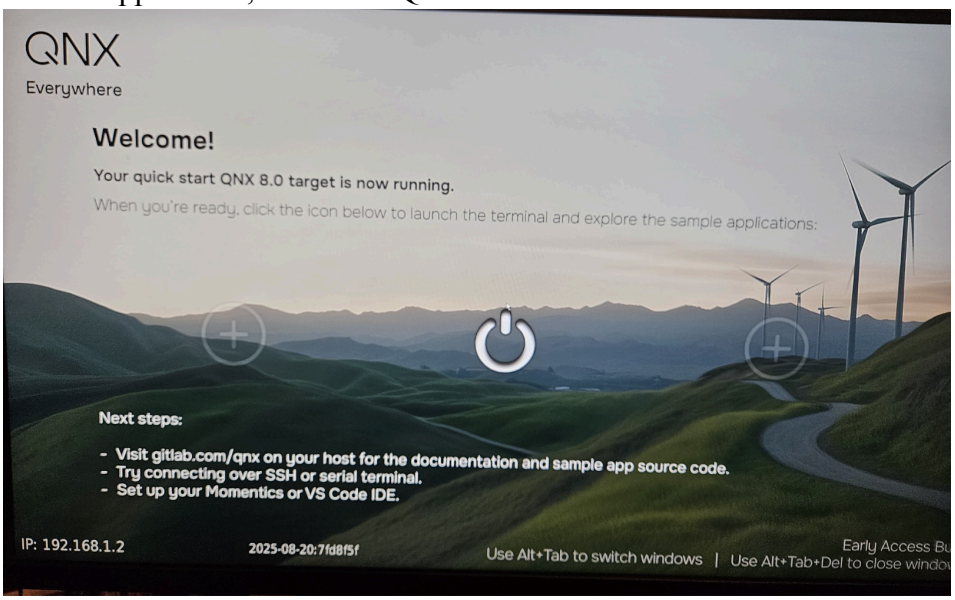

Figure 11: QNX - Ethernet Connection with IP Address Bottom Left

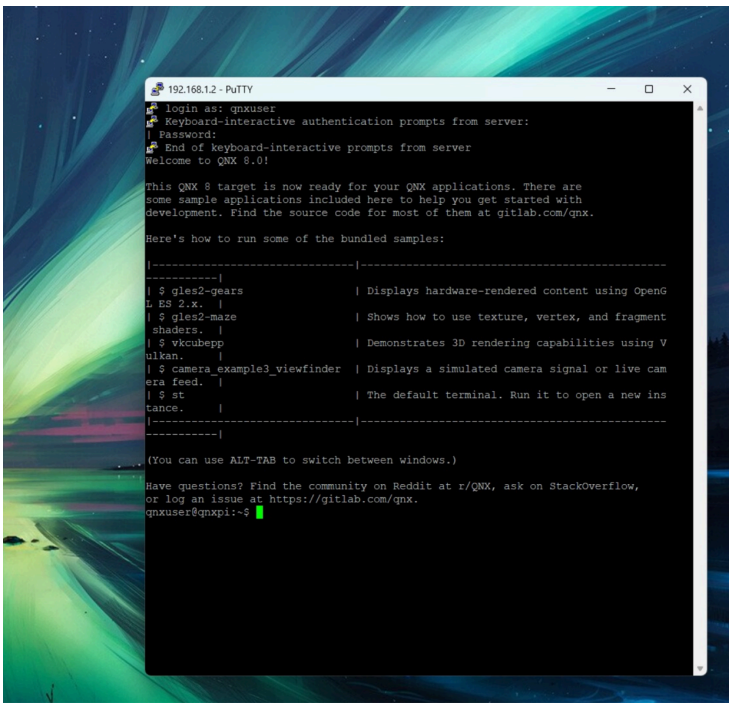

Figure 12: QNX Screen with Ethernet Cable Connection

5. Thermal temperature:

The Raspberry Pi 4 thermal temperature can be viewed in the thermal document under the dev folder. Cat the document and there one can view the devices.

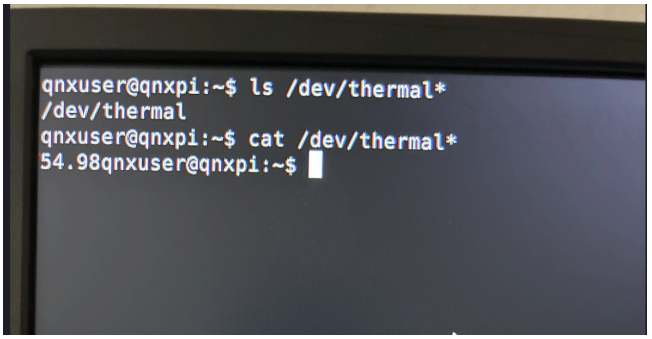

Figure13 : Thermal Screen Capture

Figure 14: Thermal Screen Capture Using Echo

6. G.711 coder/decoder:

The C code for the decoder application is attached to this document along with the generated decoded .wav files.

The decoded .wav file says:
The ship was torn apart on a sharp reef.
Sickness kept him home the 3rd week.
The box will hold 7 gifts at once.
Jazz and swing fans like fast music

7. How is the behavior of QNX different from Linux?

Linux has a monolithic kernel architecture, while QNX has a Microkernel architecture. QNX provides hard real-time guarantees with bounded interrupt latencies, while Linux provides soft real-time at best when the Linux kernel is patched with the PREEMPT_RT patch. QNX systems are statically composed with an image filesystem (IFS) and specific startup scripts. Linux boots with an initramfs, dynamic drivers, and systemd.

Appendix: Module 3 - Scripting with LINUX

Step 1 - Create executable script:

A complex bash script is created to monitor multiple aspects of the system. This script should be compatible with the Raspberry Pi OS system.

The functions the team generated to monitor the system are listed below.

1. Show Running Processes
2. Show Kernel Info
3. Show Disk Usage
4. Show System Uptime
5. Memory Summary
6. CPU Summary
7. Top 5 CPU Processes
8. Top 5 Memory Processes
9. Network Interfaces
10. Disk Usage Percentage
11. CPU Health Check
12. Memory Health Check
13. System Bottleneck Detector
14. Network Health Check
15. Long Running Processes
16. CPU Temperature
17. CPU Clock Speed

Step 2 - Convert script to an executable file:

The bash script is made into an executable using the chmod command. This changes the file permissions and adds the permission of making the file/script executable

```
chmod +x [name of file/script]
```

Step 3 - Run :

From the folder that contains the file, run the executable with the step below

```
./[name of the file/script]
```

Terminal Results:

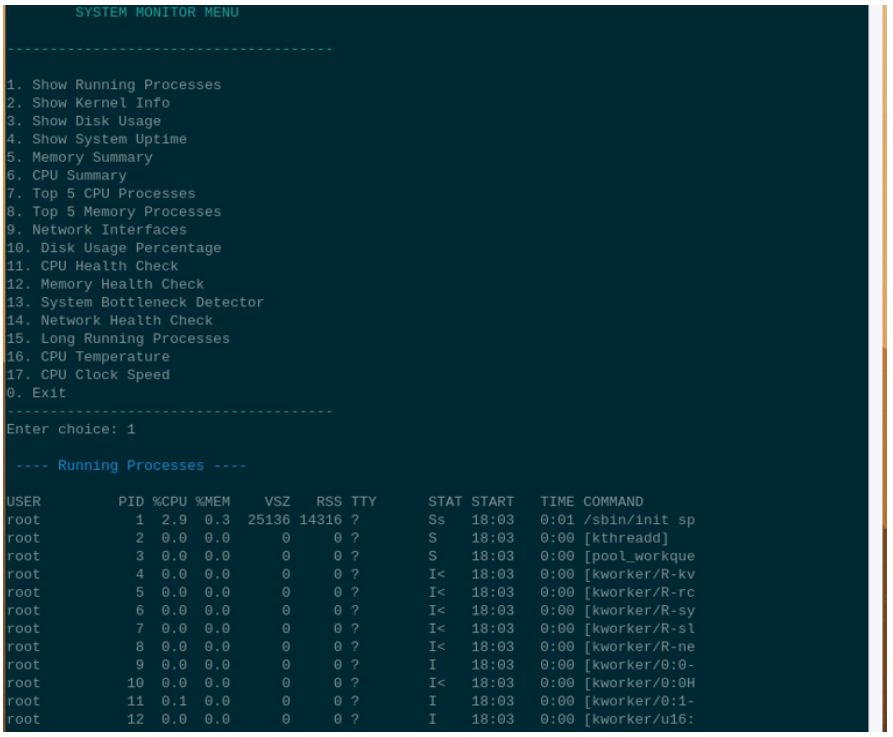

Figure 15: The executable running on Linux Terminal

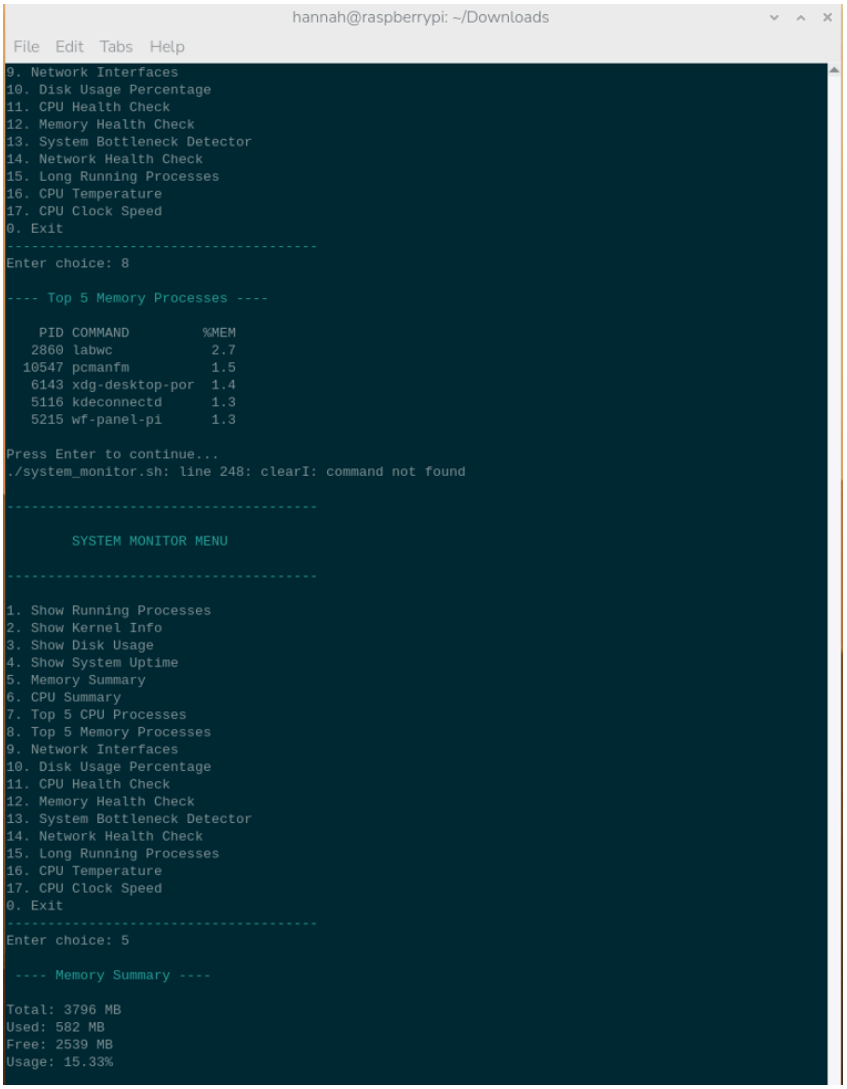

Figure 16: Example of functions in the script

Execution at Boot Time:

To run the script at boot time, a user needs to utilize `systemd` or `crontab` to execute the script. The team added the following line to the crontab file to execute the script and output a log file.

```
@reboot /bin/bash /home/hannah/Downloads/system_monitor.sh >>
/home/hannah/bootlog.txt 2>&1
```

For the script to behave the same at boot time as it does in the terminal, edits were necessary. The boot environment behaves differently from a standard terminal. During boot, the script needs to pause so the user can enter commands; otherwise, the boot process will continue, and the script will run in the background without notifying the user. There were also noticeable differences in formatting: color assignments are ignored, and special characters should be removed. Therefore, if a future client wishes to have a script that is usable in both environments, the code must account for these requirements during development

Recording Differences:

Function Name	**Boot Time**	**Terminal**
System Uptime	`16:45:54 up 0 min,  0 users,  load average: 2.08, 0.43, 0.14`	17:11:28 up 25 min, 2 users, load average: 0.77, 0.88, 0.67

Memory Summary	`Total: 3796 MB`
	`Used: 254 MB`
	`Free: 3459 MB`
	`Usage: 6.69%`
		Total: 3796 MB
		Used: 995 MB
		Free: 1552 MB
		Usage: 26.21%

CPU Summary	`User: 43.2%  System: 34.6%  Idle: 3.7%`	User: 6.1% System: 2.0% Idle: 91.8%

Top 5 CPU Processes	`PID COMMAND %CPU`
	`815 systemd-hostnam 52.1`
	`336 cloud-init 37.3`
	`694 polkitd 19.2`
	`829 ModemManager 18.7`
	`699 systemd-logind 15.2`
		PID COMMAND %CPU
		3157 ps 400
		1099 labwc 5.0
		1587 mousepad 5.0
		1965 chromium 4.5
		1871 chromium 1.9

Top 5 Memory Processes	`PID COMMAND %MEM`
	`336 cloud-init 1.2`
	`170 plymouthd 0.7`
	`1 systemd 0.3`
	`370 systemd-udevd 0.2`
	`423 (udev-worker) 0.2`
		PID COMMAND %MEM
		1965 chromium 8.2
		1810 chromium 6.9
		1871 chromium 3.8
		1099 labwc 3.1
		1974 chromium 2.6

Network Interfaces	`lo UNKNOWN 127.0.0.1/8 ::1/128`
	`eth0 DOWN`
	`wlan0 DOWN`
		lo UNKNOWN
		127.0.0.1/8 ::1/128
		eth0 UP
		192.168.1.2/24
		fe80::90c:90e9:cb84:e2a4/64
		wlan0 UP
		192.168.1.103/24
		2603:9001:5500:e9fc::1e5c/128
		2603:9001:5500:e9fc:162c:9bf2:ef69:db8b/64
		fd00:be96:e584:c451:809c:c6e5:5b0f:8a90/64
		fe80::f717:a854:3ecd:aa62/64

Disk Usage Percentage	`Filesystem Use%`
	`udev 0%`
	`tmpfs 2%`
		Filesystem Use%
		udev 0%
		tmpfs 2%

	`/dev/mmcblk0p2 15%`
	`tmpfs 0%`
	`tmpfs 1%`
	`tmpfs 0%`
	`tmpfs 0%`
	`/dev/mmcblk0p1 16%`
		/dev/mmcblk0p2 15%
		tmpfs 1%
		tmpfs 1%
		tmpfs 0%
		tmpfs 6%
		/dev/mmcblk0p1 16%
		tmpfs 1%
		tmpfs 0%

CPU Health Check	`CPU Used: [33m80.0%[0m`
	
	`Status: [33mWARNING[0m`
		CPU Used: 6.7%
		
		Status: HEALTHY

Memory Health Check	`Memory Usage: [33m6.00%[0m` `Status: [32mHEALTHY[0m`	Memory Usage: 26.00% Status: HEALTHY

System Bottleneck Detector	`CPU Idle: 47.8%`
	`Free Memory: 3413 MB`
	`IO Wait: 0.0%`
	`Bottleneck: NONE`
		CPU Idle: 91.7%
		Free Memory: 1549 MB
		IO Wait: 0.0%
		Bottleneck: NONE

Network Health Check	`Established Connections: 0`
	`Waiting Connections: 0`
	`Status: [32mNORMAL[0m`
		Established Connections: 2
		Waiting Connections: 0
		Status: NORMAL

CPU Temperature	`Temperature: 48.1 °C`	Temperature: 45.2 °C
CPU Clock Speed	`CPU Frequency: 1800.00 MHz`	CPU Frequency: 600.00 MHz

Table 1:Comparison of script at boot and at idle

Appendix: Module 4 - Build YOUR OWN PBX with Asterisk

VoIP, or Voice over Internet Protocol, is a technology that transmits voice communications over the internet instead of traditional phone lines. Asterisk is a popular open-source software framework that turns a computer into a VoIP private branch exchange (PBX), enabling it to manage phone calls, voicemail, and other telephony features. The steps below show how the team configured Asterisk to receive phone calls.

Step 1: Install Asterisk

Steps to install Asterisk onto Raspbian OS:

1. Extract the .tar file into a directory
    - `Sudo tar -xvf asterisk-22.7.0.tar.gz ; cd asterisk-22.7.0`
2. Run the configure script
    - `./configure`
3. Then build the Asterisk .tar file using make
    - `Sudo make -j4`
4. Continue with the command make for the installs, samples, and config.
    - `Sudo make install`
        i. Installs binaries into /usr/sbin

- Sudo make samples
    i. Installs example config into /etc/asterisk
- Sudo make config
    i. Enables Asterisks as a system Service

5. Enable/start Asterisk as a system service
    - Sudo systemctl enable asterisk
    - Sudo systemctl start asterisk

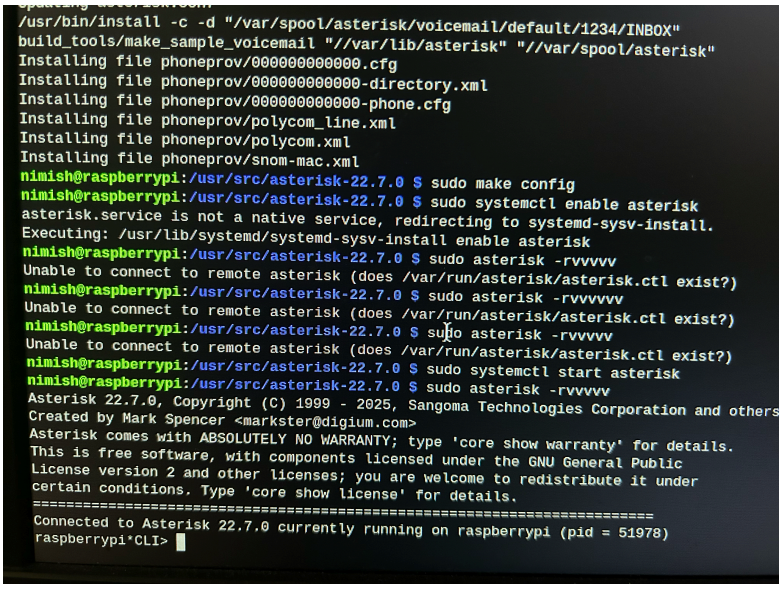

Figure 17:Asterik setup

Then, using the commands below, the memory size of the Asterisk files can be seen.

- Sudo du -sh /usr/bin/asterisk
- Sudo du -sh /usr/lib/asterisk
- Sudo du -sh /etc/asterisk

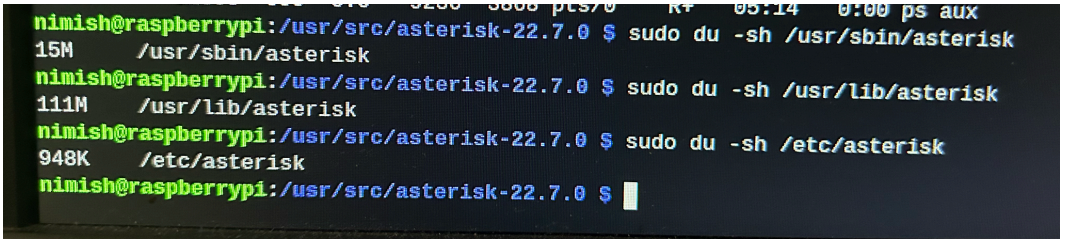

Figure 18: Memory Size

Step 2: Configure Voicemail

Zoiper was used as a softphone client, and voicemail was configured for extension 100. This required updates to pjsip.conf, voicemail.conf, and extensions.conf.

In pjsip.conf, a transport object was created to allow Asterisk to handle SIP traffic over UDP on the designated port. AOR (Address of Record) entries were added to specify the user's location and define max_contacts, which determines how many devices can register. Authentication parameters were then added, followed by the endpoint definition used by Asterisk for SIP communication.

In voicemail.conf, a voicemail mailbox was added for extension 100. In extensions.conf, a dialplan entry for extension 100 was created to ensure proper call routing. With these configurations in place, Zoiper can dial 100 and access the associated voicemail greeting.

1. Outside the asterisk CLI, perform these steps to configure Asterisk for voicemail.
   - Open extensions.conf and edit the file to set the extension value and what happens in the extension. The line below opens the file.
     i. `Sudo nano /etc/asterisk/extensions.conf`
     ii. Extension.conf changes:

```
[default]
exten => 100,1,Answer()      > answers incoming call at extension 100
  same => n,Wait(1)          > delay of 1 sec
  same => n,Voicemail(100@default)
                             > sends caller to voicemail box 100
  same => n,Hangup()         > hangsup after voicemail
```

   - Open pjsip.conf and edit the file to configure the SIP endpoint. Defining the transport, authentication, and endpoint allows Asterisk to receive a telephone call. The line below opens the file.
     i. `Sudo nano /etc/asterisk/pjsip.conf`
     ii. pjsip.conf changes:

```
[transport-udp]          > name of the transport profile
type=transport           > Tells Asterisk this SIP transport layer
protocol=udp             > SIP will use UDP
bind=0.0.0.0:5060        > listen SIP traffic here
[7001]                   > start of setting for SIP user 7001
type=aor                 > declares this block stores contact info for the
user
max_contacts=1           > only one device
[7001]                   > Auth settings for 7001
type=auth                > block handles password auth
auth_type=userpass       > username + password auth
username=7001            > SIP login username
password=7001            > SIP login password
[7001]                   > endpoint definition label
type=endpoint            > defines SIP devices' capabilities
aors=7001                > link this endpoint to AOR
auth=7001                > use auth block
context=default          > calls enter [default] from  extensions.conf
disallow=all             > disallow all audio codecs first
allow=ulaw               > enable ulaw audio codec
```

   - Open voicemail.conf and edit the file to create a voicemail mailbox for extension 100 to store messages. The line below opens the file.
     i. `Sudo nano /etc/asterisk/voicemail.conf`
     ii. voicemail.conf changes:

```
100 => 1234,Lab user,lab@example.com > Defines mailbox 100 with
pin and assigns the lab user + email
```

   - Note: Outside asterisk CLI, use sudo systemctl restart asterisk after changes have been implemented

2. Get into the CLI by opening Asterisk CLI
    - `Sudo asterisk -rvvvvv` (verbose level up to the user)
3. Continuing in Asterisk CLI, reload the Voicemail and Dialplan after configuring .conf files
    - `Voicemail reload`
    - `Dialplan reload`
4. Create an account on Zoiper with the same username and password, and give the domain as the IP of the Raspberry Pi and set the protocol to SIP.

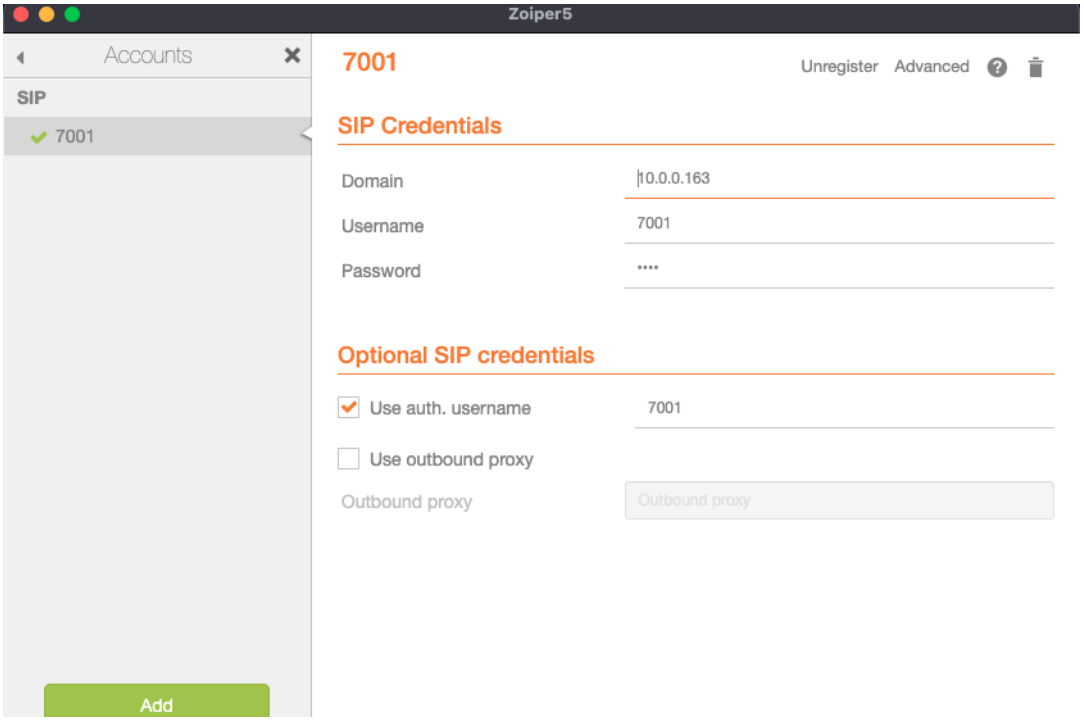

Figure 19: Zoiper Menu

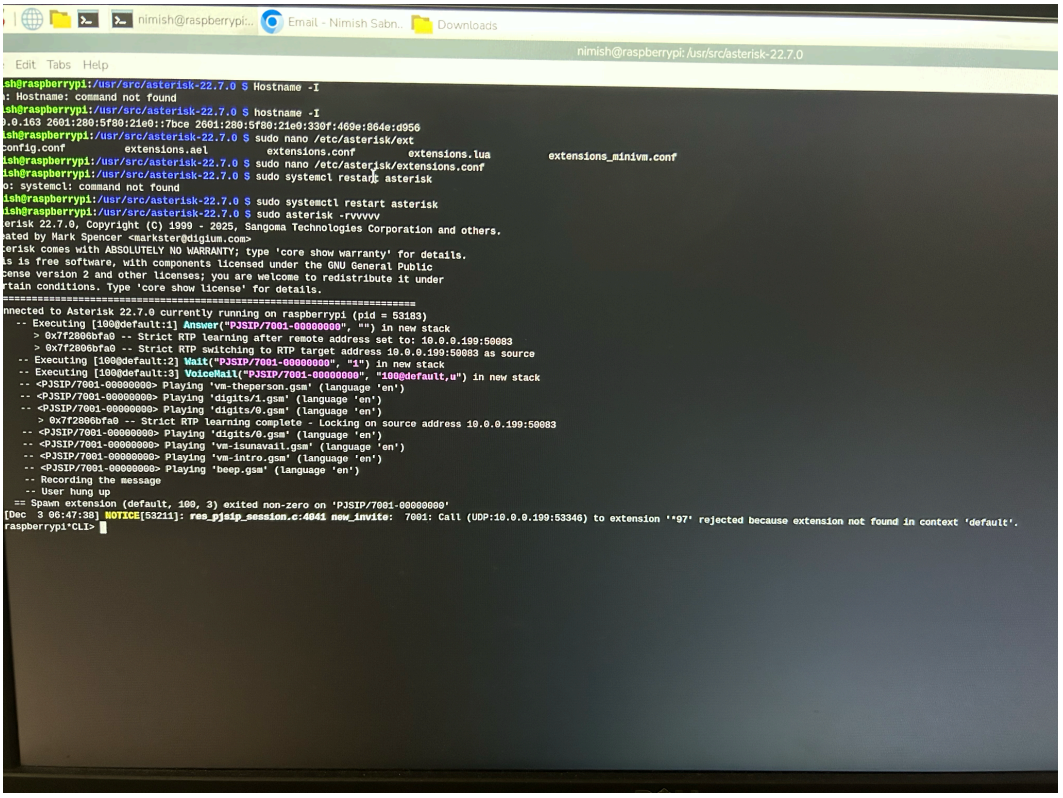

Figure 20: Asterisk Setup

Step 3: Test Phone Call

The phone call test can be seen and heard in the MP4 file within the .zip.

Appendix: Module 5 - IOT Application using QNX

We implemented a fully web-based IoT telemetry service running on QNX on a Raspberry Pi 4. The purpose of this application is to demonstrate how an embedded QNX device can expose system information over a lightweight HTTP interface mirroring how real IoT nodes publish device health, diagnostics, and sensor data to remote clients. The application launches a custom HTTP server that defines several REST-style endpoints, each accessible using standard web tools such as curl. Through these endpoints, clients can remotely retrieve live CPU usage, memory status, system uptime, and a rolling access log, all formatted as plain text. Every request triggers a timestamped entry in the server's internal log, showing how IoT devices track remote interactions. It exposes operational metrics through standard protocols rather than relying on local interaction. The complete implementation is built around QNX utilities like PIDIN and Uptime and is provided in the Python script used for deployment.

5.1 System Data Acquisition :
This part collects the live system metrics from the QNX OS. It wraps native QNX utilities like pidin cpu , pidin mem and uptime and returns the output in a sanitised way.

It invokes the commands using os.popen()- The idea behind this layer is to mimic the sensor acquisition stage in a way in actual IOT systems, except the 'sensors' here are the OS diagnostics.
5.2 Event Logging and Telemetry History
This application has a rolling access log that records every incoming HTTP request.

It timestamps each request using QNX system time
Stores human readable records like : [2025-12-04 18:33:55] Accessed /metric/cpu
Enforces memory constraints by bounding the log size to max 20 entries

5.3HTTP Routing and Endpoint Dispatch
This is implemented using a subclass of http.server.SimpleHTTPRequestHandler which handles network requests and maps URLs to logical telemetry functions.

Each Endpoint is a REST like IoT Resource

Endpoint	Function
/	Home page listing available telemetry resources
/ping	Liveness / heartbeat check
/status	Combined system status (CPU + memory + uptime)

/metric/cpu	Raw CPU diagnostics
/metric/memory	Memory state report
/metric/uptime	Device uptime
/logs	Complete access log

Table 2:Directories and what it stores

5.4 Response Formatting and Transmission

Instead of returning HTML (which curl would display poorly), the application uses plain-text responses to ensure clean formatting in terminal-based clients.

Functions here:

1)Set HTTP headers manually (Content-Type: text/plain) 2)Write the prepared telemetry payload directly to the socket 3)Guarantee each response ends with a newline for proper curl rendering.

This mirrors IoT devices that expose JSON, XML intended for machine consumption.

5.5 Lightweight Embedded Web Server Module

The entire service runs on top of a single threaded socketserver.TCPServer instance

The functions it does is:

Listens on TCP port 8080
Accepts incoming HTTP GET requests
Dispatches each request to the routing module
Runs indefinitely until terminated - emulating a background IOT daemon

Because QNX lacks many high level libraries , choosing Python's built -in TCP /web server provides a minimal , portable solution compatible with QNX's userspace environment.

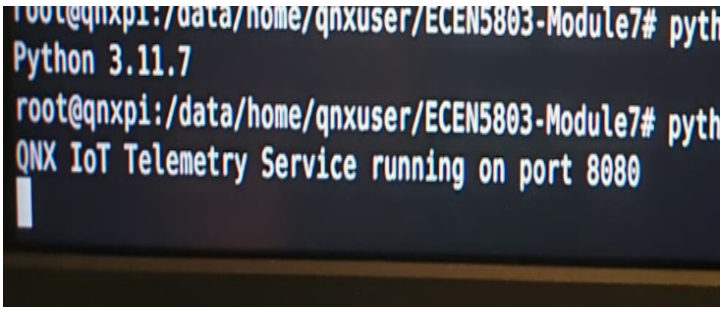

Figure 21: Screenshot of HTTP Server end on Port 8080

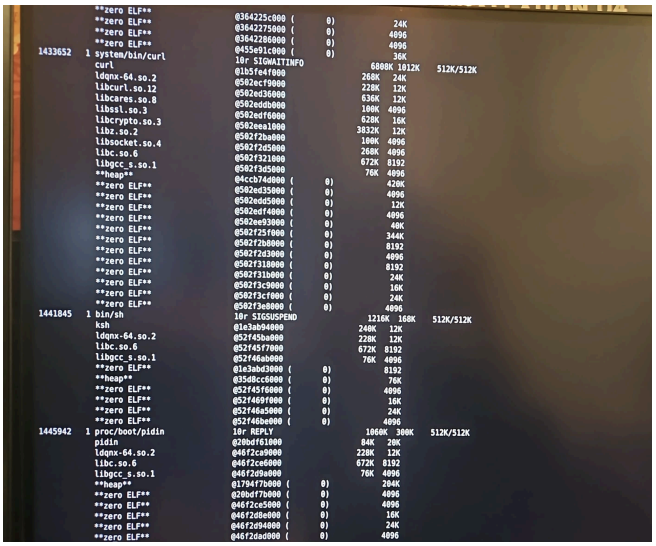

Figure 22: Screenshot of runtime memory

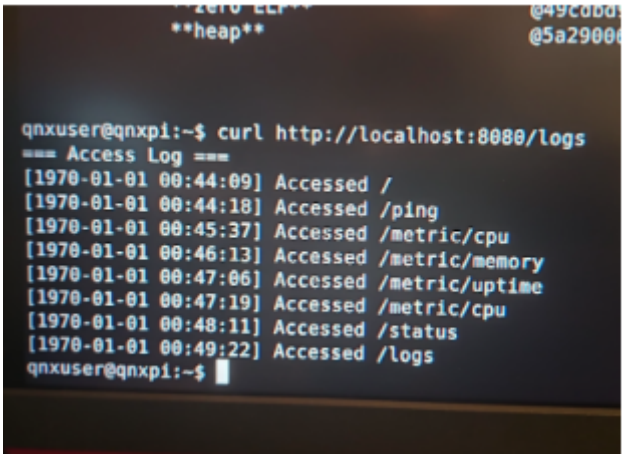

Figure 23 : Screenshot of Access Logs

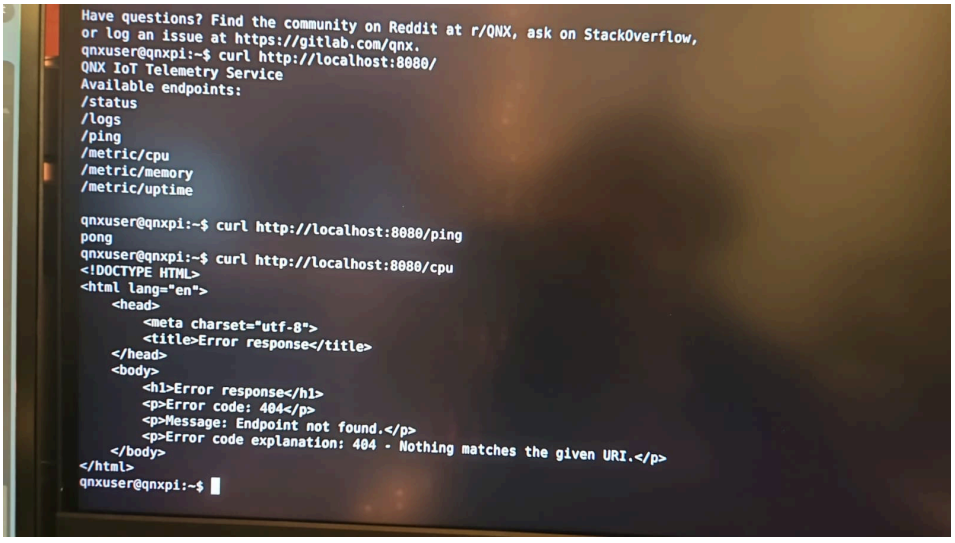

Figure 24 : Screenshot of menu for the IOT program

Appendix: Module 7 - Scripting with QNX

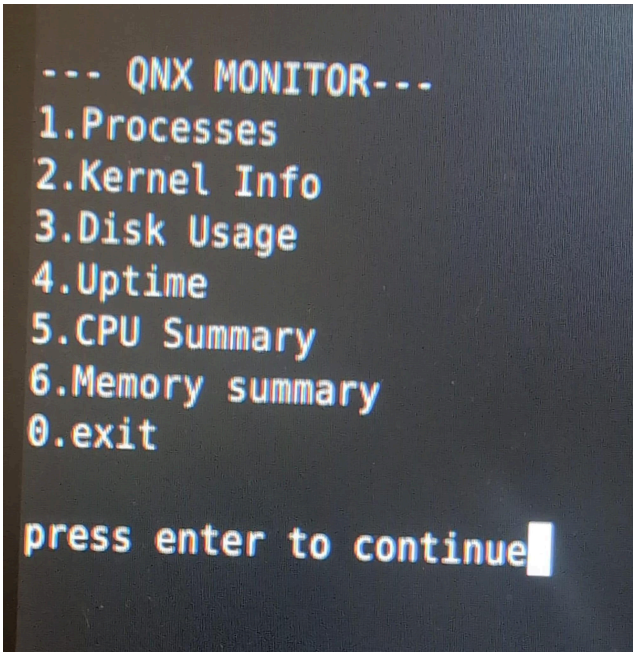

Figure 25: Screenshot of python script running on QNX

This Python script prints an interactive menu and runs QNX-native inspection commands based on the user's selection. The script uses ps -A to display active processes, uname -a for kernel/system identification, df -h for storage utilization, and uptime for load information. For platform-specific metrics, it calls QNX's diagnostic utilities pidin cpu and pidin mem, which provide CPU scheduling statistics and memory usage summaries. These commands are supported directly by the QNX Neutrino RTOS and allow the Python script to mimic a basic system status dashboard without requiring any additional packages or GUI tools.

Appendix: Bill of Materials

Part Description	Manufacturer	Manufacturer part Number	Vendor	Vendor part Number	Qty	Unit cost	Total Cost
MPU	Broadcom	BCM4908A0KFEBG	Digikey	516-BCM4908A0KFEBG-NI	1	24.86	24.86
storage	Micron Technology Inc.	MT29F1G08ABAEAWP-IT:E TI	Digikey	557-1658-2-ND	1	2.93	2.93
RJ45	TE connectivity	1-1734264-1	Digikey	A130900-ND	3	0.98	2.94
Switch interface	E-Switch	RA1113112R	Digikey	EG5619-ND	1	0.71	0.71
usb interface connector	Amphenol ICC (FCI)	87583-3010RPLF	Digikey	609-6260-2-ND	1	1.02	1.02
HDMI/Monitor Output connector	Amphenol ICC (FCI)	10029449-001RLF	Digikey	609-1010-2-ND	1	1.96	1.96
LED Indicators	Kingbright	WP59SURKSGW	Digikey	754-1548-ND	2	0.51	1.02
Power module (12 v to 5v DC/DC	Traco Power	TEA 1-0505	Digikey	1951-TEA1-0505-ND	1	1.65	1.65
Audio jack port	Kycon, Inc.	STX-3000	Digikey	2092-STX-3000-ND	1	0.64	0.64
Total						35.26	37.7

Figure 26: Image of BOM xlsx sheet

The total BOM came up to $37.7 which is within the budget. BOM excel sheet can be found with the other deliverables in the zip file.

Appendix: References

1. M. Sweetgall, “Using the G711 standard”, Code Project.com https://www.codeproject.com/articles/Using-the-G711-standard#comments-section (accessed Nov. 13, 2025)
2. “PCM A-law and u-law Companding Algorithms in ANSI C”. dystopiancode.blogspot.com. https://dystopiancode.blogspot.com/2012/02/pcm-law-and-u-law-companding-algorithms.html (accessed Nov. 13, 2025)
3. Kloudvm. “Simple Bash Script to Monitor CPU, Memory and Disk Usage on Linux in 10 Lines of Code” .kloudvm.medium.com. https://kloudvm.medium.com/simple-bash-script-to-monitor-cpu-memory-and-disk-usage-on-linux-in-10-lines-of-code-e4819fe38bf1 (accessed Nov. 16, 2015).
4. L. Rendek, “Bash script to monitor CPU and Memory usage on Linux”, linuxconfig.org. https://linuxconfig.org/bash-script-to-monitor-cpu-and-memory-usage-on-linux (accessed Nov. 16, 2015).
5. J. Ellingwood and A. Singh Walia, “Manage Systemd Services with systemctl on Linux”, DigitalOcean.com. https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-services-and-units (accessed Nov. 16, 2015).
6. Yogesh. (2014, October 17) “Get the load, cpu usage and time of executing a bash script”. stackoverflow.com. https://stackoverflow.com/questions/26425154/get-the-load-cpu-usage-and-time-of-executing-a-bash-script (accessed Nov. 16, 2015).
7. “BCM49408”. www.broadcom.com. BCM49408 | 64 bit Quad-Core ARM v8 compliant Processor (accessed Dec. 5, 2025)

Appendix: Project Team Staffing

Nimish Sabnis
MS Student at University of Colorado Boulder
Email: nisa8867@colorado.edu

Nimisha Madapura Rajashekhar
MS Student at University of Colorado Boulder
Email: nima3072@colorado.edu

Hannah Caldwell-Meurer
MS Student at University of Colorado Boulder
Email: haca8627@colorado.edu