ECEN 5803 Mastering Embedded System Architecture

Project 2
"VoIP Gateway Design Evaluation"

Date: December 15th 2023

Authors
Kiran Jojare
Viraj Patel

Professor
Timothy Scherr

Table of Contents

1. Executive Summary

The "VoIP Gateway Design Evaluation" project embarked on a series of comprehensive phases aimed at developing and accessing a VoIP Gateway on the Broadcom BCM2837 processor. The initial phase was dedicated to setting up a Windows CE build environment, crucial for creating custom Windows Embedded Compact 7 builds. This was followed by the successful booting of the BCM2837 on Raspberry Pi 3 B with Windows 10 IoT Core, testing its compatibility with Windows's IoT Core platform. Subsequent efforts involved scripting in Linux, with scripts configured to execute at start-up, showcasing the BCM2837's flexibility with different operating systems. A significant achievement of the project was constructing a PBX system using Asterisk over BCM2837 on Raspberry Pi 3 B, demonstrating its practical application in real-world telecom scenarios. Additionally, a Morse Code decoder and encoder application was developed using Python, underlining the BCM2837 on Raspberry Pi 3 B's capability in handling complex telecom tasks. The project concluded with an extensive DMIPS benchmarking phase, where the performance of the BCM2837 under Linux was evaluated across various core configurations.

Each phase of the project yielded significant results, contributing to a holistic understanding of the Broadcom processor's capabilities. Phase 1 successfully established the Windows CE build environment, setting the stage for further development. Phase 2's accomplishments included not only the efficient booting of Windows 10 IoT on the BCM2837 but also the effective implementation of the G.711 coder/decoder, which demonstrated the processor's ability to handle audio data processing. In Phase 3, the Python script developed on Raspberry Pi OS provided in-depth insights into the processor's performance, particularly highlighting its strengths in system monitoring and process management. The construction of a PBX system using Asterisk in Phase 4 underscored the processor's capability in telecommunications protocol management. Phase 5's development of a Morse Code encoder and decoder showcased the processor's versatility in IoT applications. Phase 6 was particularly noteworthy for the creation of a custom Windows Embedded Compact 7 OS, highlighting the processor's adaptability in different operating environments. Finally, Phase 7's benchmarking provided concrete data, with the processor achieving DMIPS scores of 3614.25 for quad-core configurations, respectively, showcasing its robustness and efficiency in handling varied computational demands.

The Broadcom BCM2837 processor is highly recommended for VoIP gateway development, displaying versatility with Windows 10 IoT Core and Linux. Its effective performance in system analysis, telecom protocol management, and IoT applications demonstrates its potential in various domains. Future work could further optimize this processor in different operating systems and expand telecom and IoT applications. The project's diverse phases and significant performance benchmarks firmly position the MPU as a robust solution for VoIP applications. In conclusion, based on the project's outcomes, the decision is a definitive **'GO'** for using the Broadcom BCM2837 processor in VoIP gateway applications.

2. Problem Statement and Objectives

The "VoIP Gateway Design Evaluation" project focuses on assessing the Broadcom BCM2837 processor to determine its suitability for VoIP applications. The challenge lies in evaluating whether this processor can meet the rigorous demands of VoIP technology. The project aims to address the critical need for a comprehensive hardware and software evaluation of the Broadcom processor to ascertain its efficiency and performance in the context of VoIP gateways.

The primary objectives of this project encompass evaluating the hardware capabilities of the Broadcom BCM2837 processor, assessing its software performance in executing VoIP tasks, conducting functional testing of the VoIP gateway prototype, benchmarking its computational performance, ensuring seamless integration of system components, implementing real-time monitoring solutions, and providing clear and actionable recommendations for VoIP gateway development. The ultimate goal is to deliver valuable insights into the Broadcom BCM2837 processor's suitability for VoIP gateway applications.

3. Approach and Methodology for Evaluation

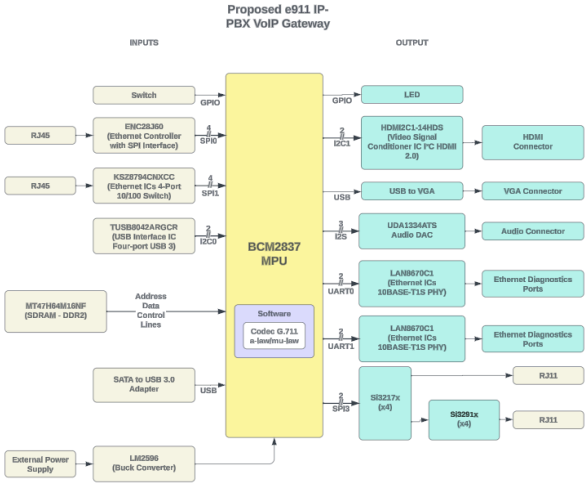

Figure 1 : Block Diagram Of Proposed e911 IP- PBX VoIP Gateway

3.1. Module 1: Create The Windows CE Build Environment

In Module 1, the team established the Windows CE build environment necessary for the VoIP Gateway Design Evaluation project. This environment, hosted within Visual Studio 2008, included key components such as .NET Compact Framework 3.5 and Windows Embedded Compact 7. Following established industry standards, the team meticulously configured the development environment, ensuring that all prerequisites were met and capturing a screenshot of the Platform Builder interface. This initiative lays the groundwork for the comprehensive evaluation and testing of the Broadcom BCM2837 processor's performance in VoIP gateway applications.

3.2. Module 2: Boot Windows 10 IoT

In Module 2 of the VoIP Gateway Design Evaluation project, the team focused on booting Windows 10 IoT on the BCM2837-driven Raspberry Pi hardware, following standard procedures provided in the project guidelines. The team exercised caution due to the early development stage of Windows 10 IoT for ARM. The team conducted a detailed evaluation of Windows 10 IoT, comparing its behaviour with that of Linux in the context of VoIP gateway applications. This comparative analysis focused on observing and documenting any distinctive behaviours and differences between the two operating systems.

The team systematically monitored the serial port for debug messages, assessed memory usage, and captured a terminal window screenshot. Verification of Ethernet and HDMI port functionality was conducted, with the option to connect to a monitor via HDMI. The team also developed C code for a G.711 coder/decoder, referencing external resources. This codec was employed to decode a designated file. Alternatively, the team explored performing this task within a Linux environment using

gcc on the target board. These observations provided valuable insights into the behaviour of Windows 10 IoT compared to Linux in the context of VoIP gateway applications.

3.3. Module 3: Scripting With Linux

In Module 3 of the VoIP Gateway Design Evaluation project, the team developed a Python script within a Raspberry Pi OS environment to extract crucial information about the Linux Kernel. This script featured a suite of functions, including real-time system monitoring, process listing, memory usage statistics, and more. Notably, the script was configured to execute automatically at system start-up, enhancing its utility for system administrators and aligning with industrial standards for efficient system management. Module 3's successful outcomes fortified the project's foundation, ensuring it met industry standards for system monitoring and management, setting the stage for the project's subsequent phases with robust tools for system administration.

3.4. Module 4: Build Your Own PBX With Asterisk

In Module 4 of the VoIP Gateway Design Evaluation project, the team ventured into the realm of building a personalized Private Branch Exchange (PBX) system using Asterisk, an open-source PBX software. The team followed detailed documentation and online guides to successfully integrate Asterisk into a Raspberry Pi OS, ensuring that it aligned with industrial standards for PBX deployment. An essential aspect of this module was assessing the memory usage by the code, providing insights into the image size and resource utilization.

Furthermore, the team configured Asterisk to offer voicemail services at extension 100, and SIP phones or softphones were set up to interact with the BCM2837-driven Raspberry Pi Model 3B running Asterisk. The configuration process and system setup were meticulously documented with screenshots, adhering to industry standards for PBX configuration and VoIP service deployment. In addition, the team explored the possibility of adding extra SIP phones or softphones to the network and conducting phone-to-phone calls, showcasing the flexibility and scalability of the system. Module 4's achievements marked a significant milestone in the project, demonstrating the team's ability to create a customized PBX system following industrial standards and paving the way for further project enhancements and integrations.

3.5. Module 5: Build a Windows 10 IoT Telecom or IoT Application

In Module 5, the team focused on developing an IoT application with the capability to decode Morse code to text and text to Morse code. This Morse code decoder and encoder application was designed to run on Raspberry Pi OS. The application allowed users to input Morse code and receive the corresponding text output, as well as input text and obtain the Morse code representation. Thorough testing and documentation were carried out to ensure the functionality of this Morse code conversion application, showcasing the team's proficiency in IoT application development following industry standards.

3.6. Module 7: Measure Cortex-A53 DMIPS Under Widows 10 IoT and Linux

In Module 7, the team exclusively used Raspberry Pi OS to assess the Cortex-A53 DMIPS performance on a BCM2837-driven Raspberry Pi Model 3B, recording DMIPS by running Dhrystone benchmarks across various core configurations. Sequential execution on four different cores produced a total of 3684.18 DMIPS, marginally outperforming the concurrent four-core run, which achieved 3614.25 DMIPS. In detailed single-core tests, the highest DMIPS reached was 922.83, while dual-core tests yielded 1844.54 DMIPS. These results highlight the system's efficiency and the MPU's technical capabilities in handling tasks across multiple operating scenarios.

4. Module Test Results

4.1. Module 1: Create The Windows CE Build Environment

In Module 1, our team proficiently completed the setup of the Windows CE build environment for Windows Embedded Compact 7 achieving all the required objectives. This involved the careful installation of Visual Studio 2008 and its Service Pack 1, along with the .NET Compact Framework 3.5, and meticulously configuring them per the provided guidelines. The successful initiation of the Platform Builder within Visual Studio 2008 was a key milestone, confirming the readiness of our development environment. Detailed documentation of the process, along with relevant screenshots, has been thoroughly compiled and is available in the Appendix and GitHub.

4.2. Module 2: Boot Windows 10 IoT

In Module 2, the team successfully set up Windows 10 IoT Core on BCM2837-driven Raspberry Pi 3B, as documented in several figures showcasing the dashboard, device portal, and home page. Serial debugging was conducted using an FTDI USB to Serial adapter, with Windows Debugger capturing system messages. The image size was noted to be **1.46 GB**. Using Windows Debugger, we observed multiple attempts to establish a connection, with a successful linkage noted on November 11th. The debugger session revealed insights into the system's ARM architecture kernel, displaying its version and instruction set capabilities. These observations provided real-time tracking of the system's progress. Further details and visual documentation of these findings are included in the Appendix and GitHub.

Additionally, HDMI and Ethernet ports were tested for functionality. The team also developed a G.711 decoder in Raspberry Pi OS, processing audio files and observing the outputs. Comparative observations between Windows 10 IoT and Linux highlighted several differences: Windows 10 IoT offers a more graphical user interface akin to the traditional Windows desktop, while Raspberry Pi OS favours a command-line interface along with GUI. System management in Windows 10 IoT employs Windows Management Instrumentation (WMI), whereas Linux uses shell scripts and native commands. Troubleshooting in Windows 10 IoT is handled through the Windows Debugger, contrasting with the command line-based diagnosis in Raspberry Pi OS. The file system structure and administrative paths also differ significantly, with Windows 10 IoT presenting a more streamlined version with a smaller installation size and fewer pre-installed applications compared to the more application-rich Raspberry Pi OS. Moreover, Windows 10 IoT Core includes a device portal for enhanced device management—a feature generally absent in Raspberry Pi OS, with Fedora's Linux being a notable exception. Screenshots and detailed observations are available in the Appendix and on GitHub.

4.3. Module 3: Scripting With Linux

In Module 3, the team executed a Python script on Raspberry Pi OS, which performed multiple functions for Linux Kernel information retrieval. The script included functions for listing processes, getting kernel names and versions, displaying kernel dump, and monitoring system performance in real time. Additionally, it provided details on disk utilization, memory consumption, active and background processes, process count per user, device IP, and memory stats. The script was also

configured to run at system start-up using crontab. Each function of the script was designed to output its results into a separate text file. These files, generated at every system start-up, provide a persistent record of the system's status at each boot. The generated files and screenshots are available in the Appendix and GitHub for reference.

4.4. Module 4: Build Your Own PBX With Asterisk

In Module 4, the project team adeptly integrated the open-source PBX software Asterisk into a BCM2837-driven Raspberry Pi 3B, creating a fully functioning PBX system. The final image size for the Raspberry Pi OS with all recommended software and Asterisk installed was 15 GB, with Asterisk itself occupying just **15 MB**. The team then successfully set up and configured a PC with MicroSIP to connect to the Asterisk network running on BCM2837-driven Raspberry Pi 3B, establishing a voicemail system accessible at extension 100. This setup was part of a comprehensive effort to demonstrate the viability of BCM2837-driven Raspberry Pi 3B as a cost-effective and compact solution for PBX systems.

Further advancement in the module included adding two SIP phones to the network named 'viraj' and 'kiran.' This allowed the team to test phone-to-phone calls within the same network, effectively simulating a real-world PBX environment. The configuration details, call logs, and settings for MicroSIP and the SIP phones are thoroughly documented in the Appendix. For a clear visual reference, screenshots of the configurations and logs, along with the relevant details of the setup, have been included in the Appendix and GitHub. A video demonstration for the same is available on GitHub.

4.5. Module 5: Build a Windows 10 IoT Telecom or IoT Application

In Module 5, the team developed a Morse Code Receiver application using Python on BCM2837-driven Raspberry Pi 3B running Raspberry Pi OS. This application skilfully translates Morse code into text and vice versa. It includes a user interface for encoding, decoding, and handling error conditions. The team thoroughly tested the application with various Morse code contents and scenarios, ensuring robust performance. Detailed results, including test cases and outcomes, are systematically documented and can be found in the Appendix and GitHub.

4.6. Module 7: Measure Cortex-A53 DMIPS Under Widows 10 IoT and Linux

In Module 7, the team embarked on a detailed analysis of the DMIPS (Dhrystone MIPS) performance of the Cortex-A53 processor under both Windows 10 IoT and Linux on a BCM2837-driven Raspberry Pi Model 3B. They adeptly ported the DMIPS benchmark code and compiled it to measure the processor's performance across single, dual, and quad-core configurations. In their rigorous testing, it was observed that sequential execution of the Dhrystone benchmark on 4 cores resulted in a total DMIPS of 3684.18, which was notably higher than the concurrent execution on all four cores, yielding 3614.25 DMIPS. This suggested a better performance through sequential execution likely due to reduced resource contention and a more optimized processing flow.

Further granularity in the benchmarks revealed that individual core performances varied slightly, with Core 1 achieving 918.48 DMIPS, Core 2 peaking at 922.83 DMIPS, Core 3 at 921.61 DMIPS, and Core 4 at 921.27 DMIPS. The highest single-core DMIPS recorded was 922.83, and when pairing two cores, the DMIPS totalled 1844.44 (Core 2 and Core 3 combined). The quad-core sequential run matched the sum of the individual core scores at 3684.18 DMIPS, confirming the accuracy of the measurements and providing valuable insights into the scalability and efficiency of the processor's performance under different operating systems. These findings are exhaustively documented in the Appendix and GitHub, offering a resource for further analysis and comparison.

5. List of Project Deliverables

The project's deliverables are meticulously organized within a comprehensive ZIP file, providing easy access to essential project components. This archive includes module-specific project files, detailed test results, individual module reports, a structured block diagram, BOM, video demonstrations and screenshots capturing various aspects of the project's functionality. For enhanced accessibility and collaboration, all these critical project elements are also accessible through the GitHub repository, ensuring a seamless and interactive experience for project stakeholders.

6. Recommendations - GO

In the VoIP Gateway Design Evaluation project, it is advisable to employ the Broadcom BCM2837 processor found in the Raspberry Pi 3B as the central component. This choice is based on a thorough evaluation of key project factors, including functionality, cost-effectiveness, development ease, and adherence to budget constraints. The Broadcom BCM2837 processor stands out as an optimal solution, given its versatility and affordability. Its compatibility with both Windows 10 IoT and Linux makes it ideal for implementation over various operating system environments. Moreover, the processor's extensive community support, along with its compatibility with various open-source software, simplifies development efforts. Importantly, the Bill of Materials (BOM) for the VoIP Gateway Design Evaluation project totals $49.975. It aligns with budgetary limitations and offers ample computing power and memory resources. Leveraging the Broadcom BCM2837 processor is recommended, as it ensures a cost-effective and feasible approach to achieving the project's objectives, warranting a **'GO'** decision for its utilization.

7. Appendix: References

[1]. Project2 Guide.pdf.
[2]. Raspberry Pi 3 B Datasheet.
[3]. Raspberry Pi 3 B Reference Manual.
[4]. Windows 10 IoT Core Documentation.
[5]. Windows Embedded Compact 7 Evaluation Documentation
[6]. Raspberry Pi 3 Model B User's Guide
[7]. Broadcom BCM2837 ARM Peripherals Guide
[8]. Linux Scripting Manuals.
[9]. Asterisk Manuals for Open Source PBX.

8. Appendix: Test Results

8.1. Module 1: Create The Windows CE Build Environment

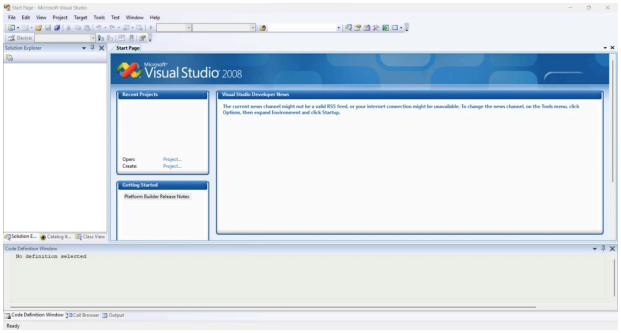

Figure 2 : Platform Builder in Visual Studio

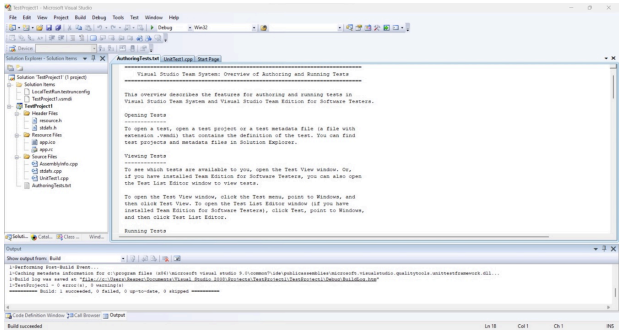

Figure 3 : Test Project

8.2. Module 2: Boot Windows 10 IoT

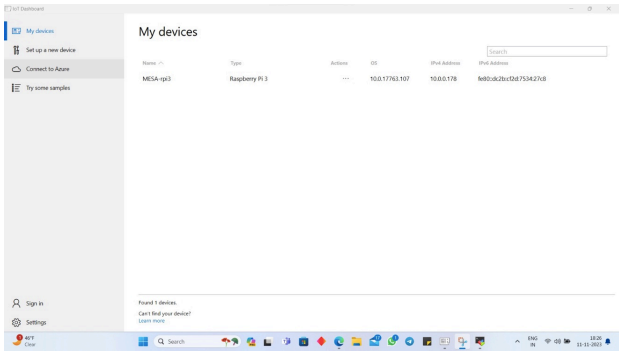

Figure 4 : Windows 10 IoT Core Dashboard

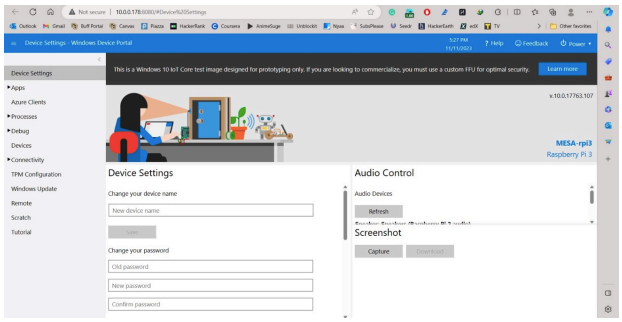

Figure 5 : Windows 10 IoT Core Device Portal

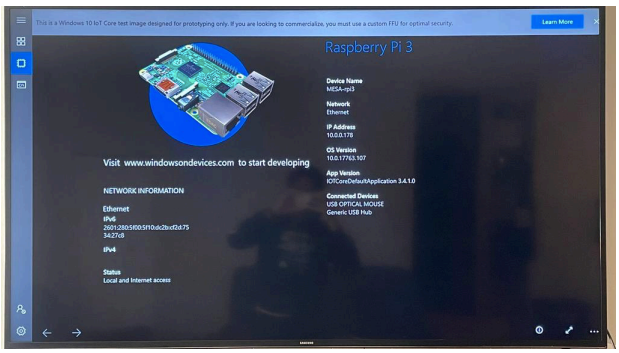

Figure 6 : Windows 10 IoT Core Home Window After Set-up

Figure 7 : FTDI Connections

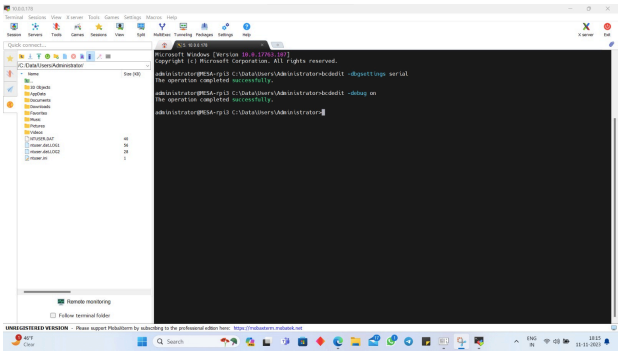

Figure 8 : Terminal Window Showing Commands to Enable Debugging

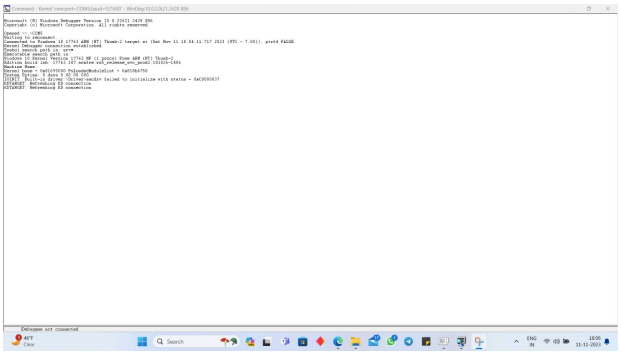

Figure 9 : Kernel Debug During Initial Boot

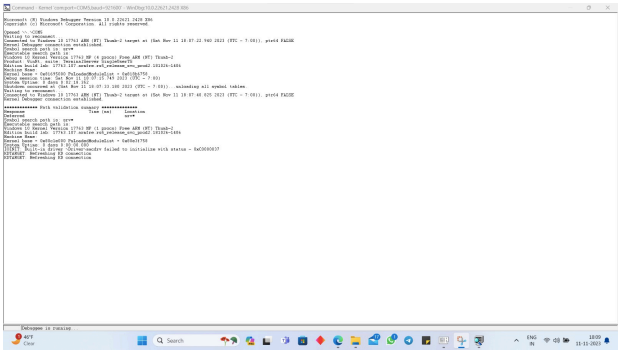

Figure 10 : Kernel Debug After Initial Boot

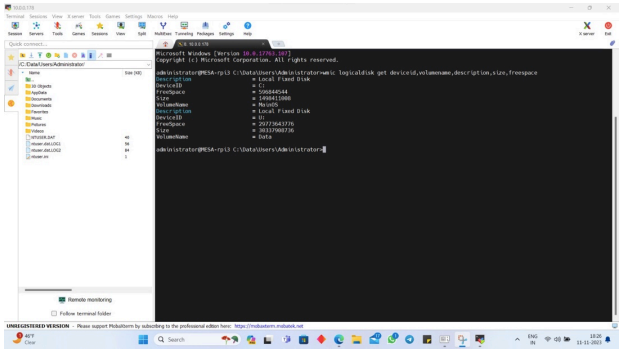

Figure 11 : Windows 10 IoT Core Image Size

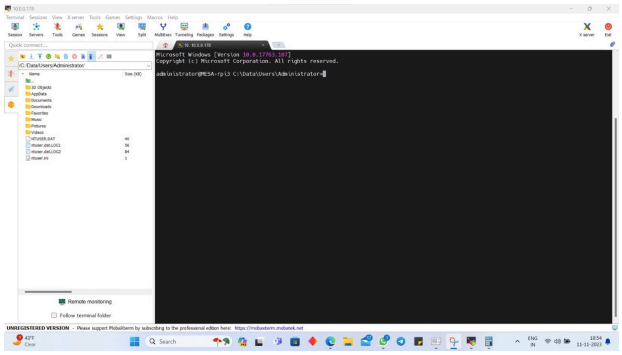

Figure 12 : Command Terminal Using SSH

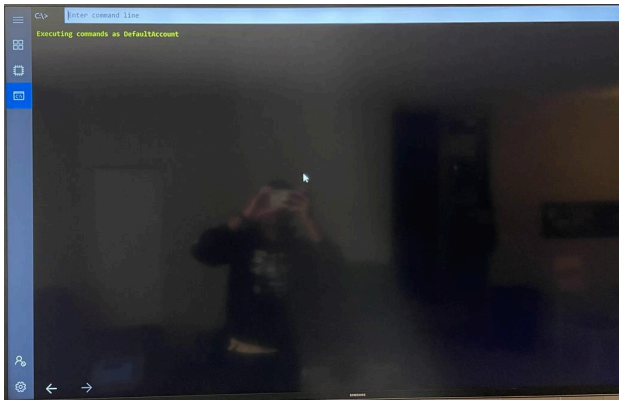

Figure 13 : Command Terminal On Raspberry PI without SSH using HDMI

8.3. Module 3: Scripting With Linux

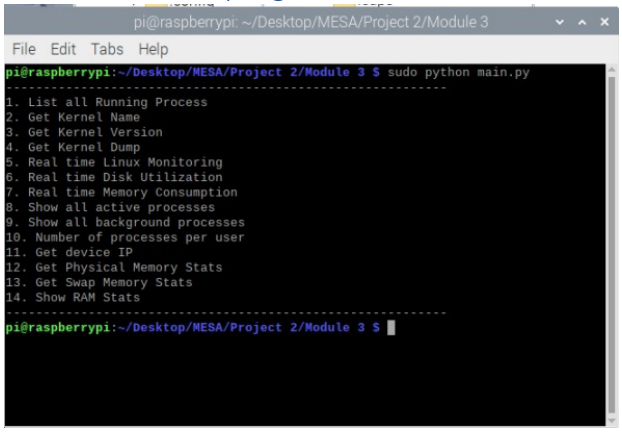

Figure 14 : Command-Line of Python Code

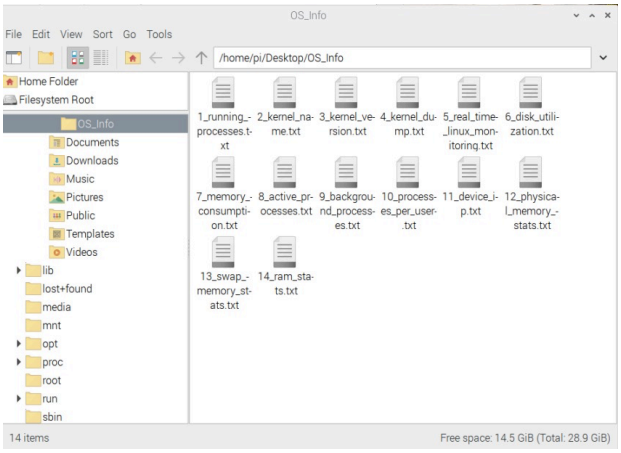

Figure 15 : Text File generated After Successful Bootup

8.4: Module 4: Build Your Own PBX With Asterisk

```
pi@raspberrypi:~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        29G   15G   14G  53% /
devtmpfs        325M     0  325M   0% /dev
tmpfs           455M     0  455M   0% /dev/shm
tmpfs           182M  4.0M  178M   3% /run
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
/dev/mmcblk0p1  255M   31M  225M  13% /boot
tmpfs            91M   52K   91M   1% /run/user/1000
pi@raspberrypi:~ $ du -h /usr/lib/asterisk
14M     /usr/lib/asterisk/modules
14M     /usr/lib/asterisk
pi@raspberrypi:~ $ du -h /etc/asterisk/
8.0K    /etc/asterisk/manager.d
1.0M    /etc/asterisk/
pi@raspberrypi:~ $
```

Figure 16 : Image Size

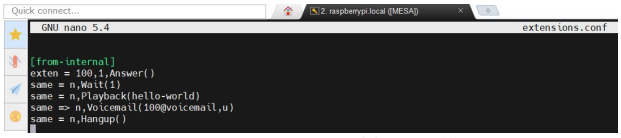

Figure 17 : extensions.conf for Dialling 100

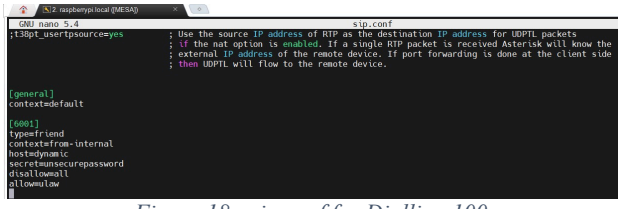

Figure 18 : sip.conf for Dialling 100

Figure 19 : voicemail.conf for Dialling 100

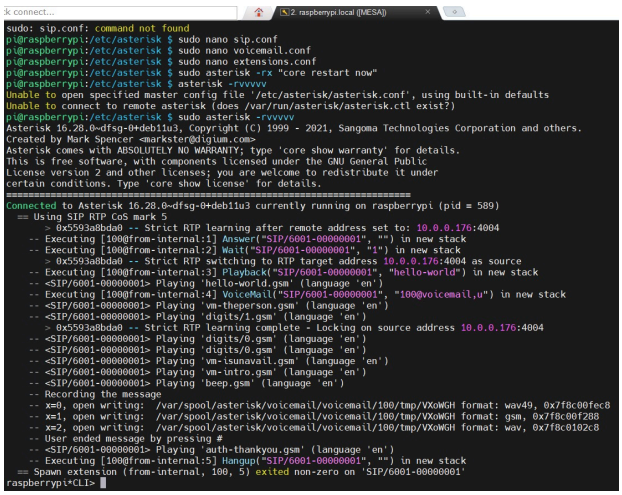

Figure 20 : Logs after dialling the extension 100 through MicroSIP

Figure 21 : extensions.conf for Phone-to-Phone Setup

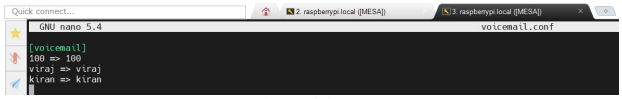

Figure 22 : voicemail.conf for Phone to Phone Setup

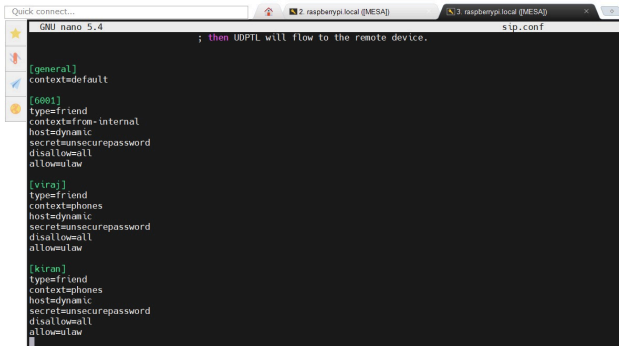

Figure 23 : sip.conf for Phone-to-Phone Setup

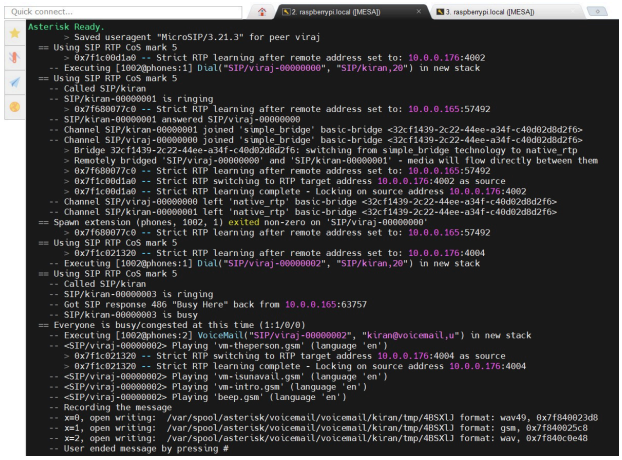

Figure 24 : Log screenshot 1 after dialling the extension 'viraj' from 'kiran' through a softphone app MicroSIP

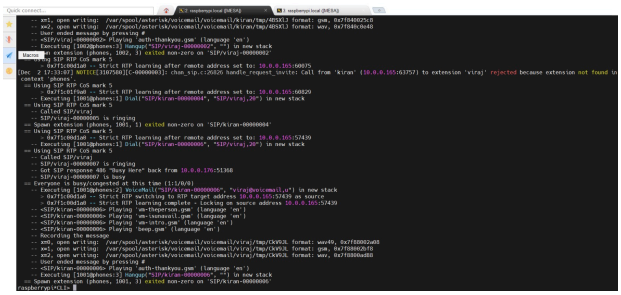

Figure 25 : Log screenshot 2 after dialling the extension 'viraj' from 'kiran' through a softphone app MicroSIP

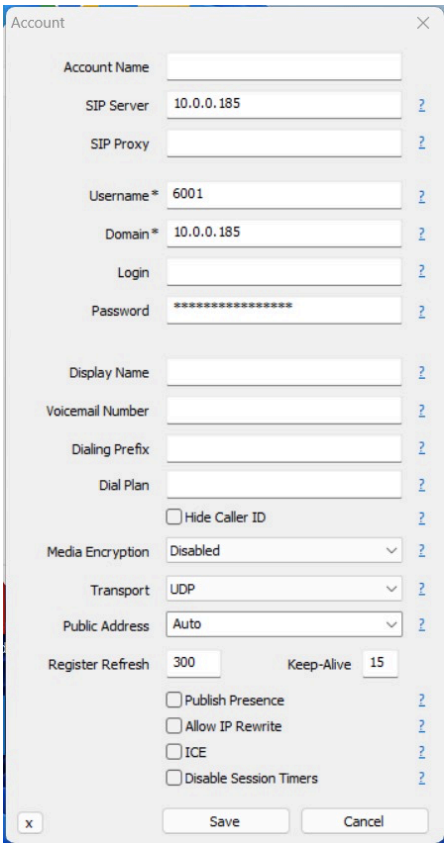

Figure 26 : MicroSIP configuration for 100 i.e "6001"

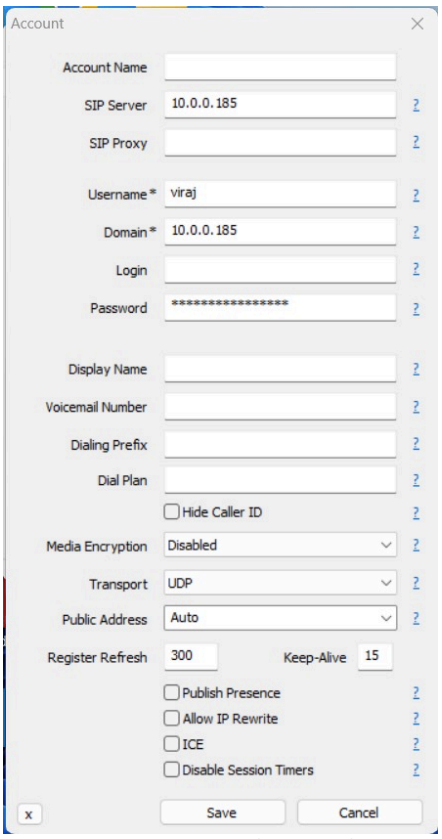

Figure 27 : MicroSIP configurator for "viraj"

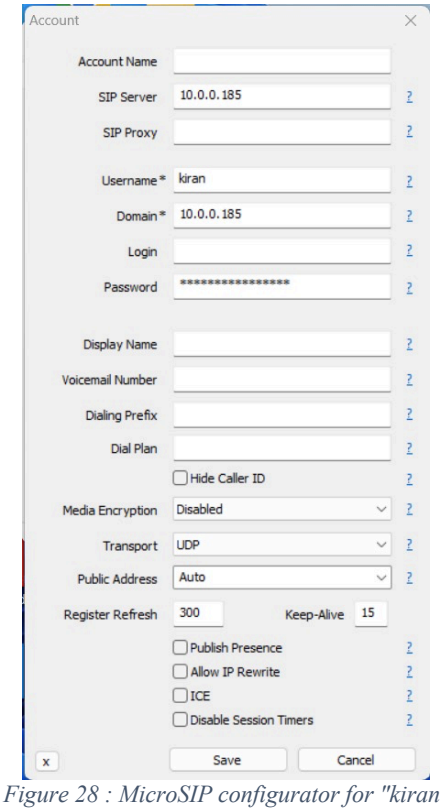

Figure 28 : MicroSIP configurator for "kiran"

8.5: Module 5: Build a Windows 10 IoT Telecom or IoT Application

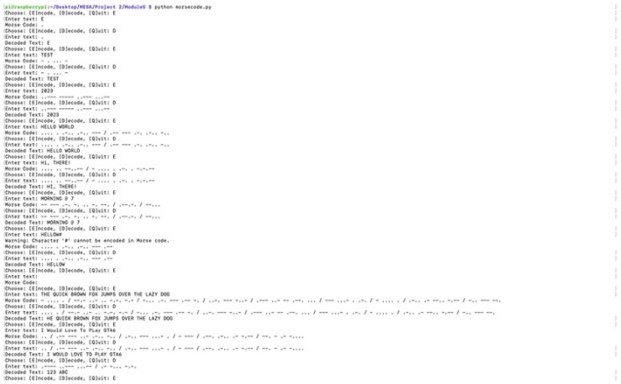

Figure 29 : Output of Morse Code Decoder/Encoder

8.6. Module 7: Measure Cortex-A53 DMIPS Under Widows 10 IoT and Linux

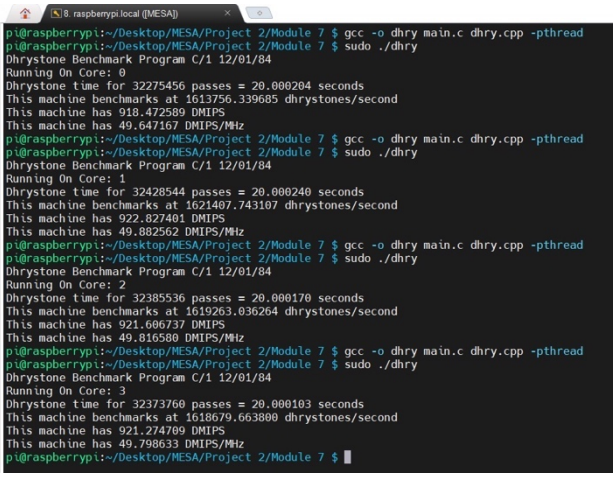

Figure 30 : Sequential Dhrystone Benchmarking on Each of the Four Cores (Linux)

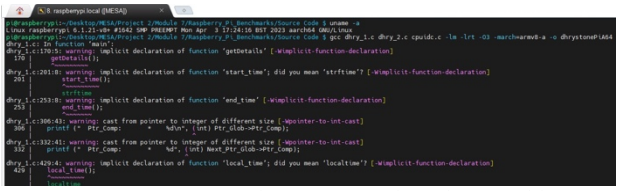

Figure 31 : Concurrent Dhrystone Benchmarking Build Across All Four Cores (Linux)

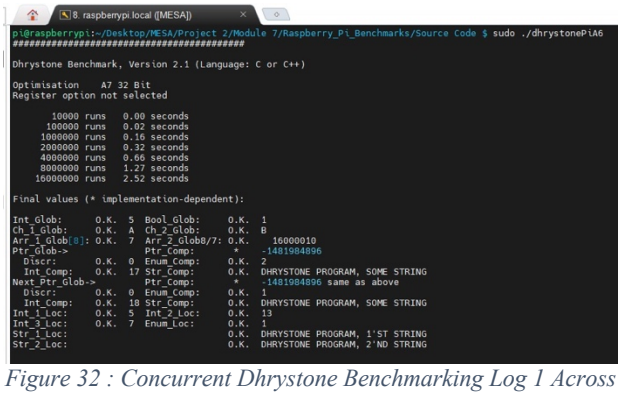

Figure 32 : Concurrent Dhrystone Benchmarking Log 1 Across All Four Cores (Linux)

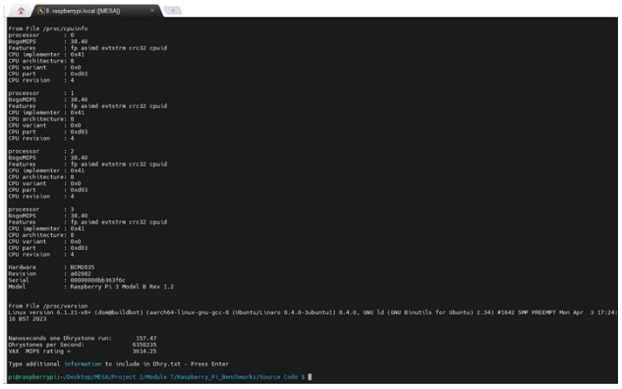

Figure 33 : Concurrent Dhrystone Benchmarking Log 2 Across All Four Cores (Linux)

9. Appendix: Frequently Asked Questions

9.1. Module 1

9.1.1. Start the Platform Builder, and capture a screenshot.

The screenshot is seen in figure(1) and figure (2) in the Module 1 test results section.

9.2. Module 2:

9.2.1. Upon booting up, the serial port should be sending out debug messages. Open a terminal window to capture them. What do you see? Be aware that the port used to transmit the information may be different from what you expect.

Figure(9) and (10) in Module 2 Screenshots shows the Windows Debugger(windbg) -> Kernel Debugger during the initial boot and after reboot respectively.

The image shown in Figure (10) displays a session of the Windows Debugger (windbg) engaged with a Windows 10 IoT Core system on ARM architecture. The WinDbg version is 10.0.22621.24028 x86, and it shows multiple attempts to establish a connection, with a successful connection logged on November 11th. The system's kernel version is 17763 for an ARM processor with Free Build and Thumb-2 instruction set, indicating it's designed for non-commercial use with advanced CPU instruction capabilities. The debugger's symbol path is set to a server, suggesting it's using remote symbols for debugging. An error message reveals a driver initialisation failure with a specific status code. The machine's name is not displayed, instead represented by a base memory address, and the system uptime is minimal, indicating a recent reboot or start-up. The output includes a detailed timestamp, reflecting the debugger's real-time tracking of the session's progress.

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

The image size of 1.46 GB is calculated from the command “wmic logicaldisk” as seen in). Refer to the calculation below for the image size calculated.

(1498411008 - 596844544) + (30337908736 - 29773643776) = 1,46,58,31,424 Bytes = **1.46 GB**.

9.2.3. What temperature is displayed on your PC terminal window? What is displayed on the LCD? The Ethernet and HDMI ports should also be active. Connect the HDMI output to a monitor using an HDMI Cable and adapter if necessary, or connect using SSH to see the GUI window. Reboot the system – what do you see?

After the initial boot setting using HDMI & Ethernet, we see the Windows 10 IoT Core home page as seen in figure(14).

9.2.4. How is the behaviour of Windows 10 IoT different from Linux?

Based on our observations through the screenshots shared:

We note that Windows 10 IoT presents a more graphical user interface that is similar to the traditional Windows desktop environment, whereas Raspberry Pi OS, which is a Linux distribution tailored for the Raspberry Pi, often emphasizes a command-line interface for operations, particularly in IoT or embedded systems.
We observe that Windows 10 IoT uses Windows Management Instrumentation (WMI) for system management, a framework unique to Windows. In contrast, in Raspberry Pi OS, we would typically use various shell scripts and native Linux commands for this purpose.

From the Windows Debugger output, we discern that Windows 10 IoT follows a different method for troubleshooting and system feedback compared to the hands-on, text-based diagnosis process we are accustomed to with Raspberry Pi OS.

The file system structure and administrative paths shown in the screenshots indicate a significant difference in system organization between Windows 10 IoT and Raspberry Pi OS.

We also note the smaller size of the Windows 10 IoT installation compared to Raspberry Pi OS. This is due to Windows 10 IoT Core being a more streamlined version of Windows with fewer pre-installed applications, as opposed to Raspberry Pi OS which typically comes with a set of applications included.

9.3: Module 3

No Questions. Refer to the appendix of test results for module 3 for more details.

9.4: Module 4

No Questions. Refer to the appendix of test results for module 4 for more details.

9.5. Module 5

9.5.1. For the development of code in Linux, GCC and GDB are the preferred compilers and debuggers to use. Go to www.asterisk.org and download Asterisk. Look at https://wiki.asterisk.org/wiki/display/AST/Beginning+Asterisk and read the sections on Beginning Asterisk, Installing Asterisk, and the Hello World Project. Use either the Angstrom package repository to install the asterisk package directly after connecting the Raspberry Pi Model 3 to the internet under Linux. Carefully read the documentation and online guides and incorporate Asterisk, the open source PBX into one of your Linux SD cards. How much memory is used by the code? (What is the image size?)

We successfully finished setting up an Asterisk on our Raspberry Pi 3. Following the available guides, we also ran the 'Hello World' example provided in the documentation.

Image Size Overall = **15 GB**
Image size of Asterisk = **15 MB**

The considerable size of the total image results from a complete installation of the Raspberry Pi OS, encompassing all the recommended software.

9.6. Module 7

9.6.1. How is the behaviour of Windows 10 IoT different from Linux? Port your DMIPs benchmark code from Project 2 and compile and run it under Linux on the Raspberry Pi Model 3. Record you DMIPs numbers using 1, 2, or 4 cores. 2. Port your DMIPs benchmark code from Project 2 and compile and run it under Windows 10 IoT on the Raspberry Pi Model 3. Does it match your expectations? How does it compare to the Linux? implementation – do you get the same performance numbers?

The DMIPS benchmarking code was skilfully adapted and executed to assess the processor's throughput on single, dual, and quad-core setups. It was found that running the Dhrystone benchmark sequentially on four cores tallied a DMIPS sum of 3684.18, surpassing the 3614.25 DMIPS scored during simultaneous execution across all cores. This indicates that sequential processing may enhance performance, likely owing to lessened contention for resources and improved processing efficiency.

The benchmarks also showed minor variations in performance across the individual cores: Core 1 hit 918.48 DMIPS, Core 2 reached the highest at 922.83 DMIPS, Core 3 notched up 921.61 DMIPS, and Core 4 logged 921.27 DMIPS. The peak performance for a single core was recorded at 922.83 DMIPS, and a dual-core setup involving Core 2 and Core 3 achieved a combined score of 1844.44 DMIPS. A sequential execution on all four cores confirmed the individual core results with a collective score of 3684.18 DMIPS, underscoring the precision of the benchmarking process and shedding light on the processor's performance scale and efficiency across varied OS environments.

10. Appendix: Bill Of Material (BOM)

The Bill Of Material (BOM) called BOM.xlsx is available in a submitted folder at Final Report. The final cost falls within the range of 50$ as per requirement.

Item No.	Description	Quantity	Unit Cost ($)	Total Cost ($)	Vendor	Vendor Part Number
1	MPU	1	1.2	1.2	AliExpress	BCM2837
2	Stand-Alone Ethernet Controller	1	4.09	4.09	Microchip	ENC28J60
3	Ethernet ICs 4-Port 10/100 Switch	1	6.19	6.19	Microchip	KSZ8794CNXCC
4	RJ45 female Connector	2	0.99	1.98	Link-PP	LPJ4135GDNL
4	USB Interface IC Four-port USB 3	1	3.68	3.68	Texas Instruments	TUSB8042ARGCR
5	LED	1	0.11	0.11	DigiKey	1497-XCMDK12D-ND
6	IC VIDEO HDMI 2.0	1	1.14	1.14	Digikey	HDMI2C1-14HDS
7	HDMI Connector	1	0.43	0.43	Digikey	10118192-0002LF
8	USB to VGA Converter	1	1.74	1.74	Jakcom by AliExpress	NA
9	Audio DAC IC	1	0.685	0.685	NXP	UDA1334ATS
10	Audio Connector	1	0.45	0.45	AliExpress	PJ307 3F07
11	Ethernet PHY (Physical Layer) chip	2	2.83	5.66	Microchip	LAN8670C1-E/LMX
12	BRI Ports	4	1.13	4.52	ADI	Si3217
		4	0.99	3.96	ADI	Si3291
13	Buck Converter(12V-5V)	1	1.02	1.02	Shenzhen Yunuo Electronic	LM2596
14	DDR2 RAM 1 GB	1	4.31	4.31	Micron	MT47H64M16NF
15	SATA to USB Cable	1	2.29	2.29	EYOOLD	NA
16	PCB Assembly Cost With Connectors	15% of Total Cost		6.52		
		Total		**49.975**		

Figure 34 : Bill Of Material (BOM)

11. Appendix: Project Staffing

Kiran Jojare
Graduate Student
University of Colorado, Boulder.
Email: kijo7257@colorado.edu
Phone: +1 (720) 645 6212

Viraj Patel
Graduate Student
University of Colorado, Boulder.
Email: vipa5773@colorado.edu
Phone: +1 (720) 561 3212