Shape

Description automatically generated with medium confidence

EEET2490 – Embedded System: OS and Interfacing, Semester 2024-1

Assessment 2 – Individual Assignment Report

ADDITIONAL FEATURES FOR BARE METAL OS

Lecturer: Mr Linh Tran – [linh.tranduc@rmit.edu.vn](mailto:linh.tranduc@rmit.edu.vn)

Student name : Tran Thanh Tu

Student ID : S3957386

Date : 27/04/2024

TABLE OF CONTENTS

[I. INTRODUCTION 1](#_Toc165449714)

[II. ADDITIONAL FEATURES FOR BARE METAL OS 1](#_Toc165449715)

[1. Background 1](#_Toc165449716)

[2. Welcome message and Command Line Interpreter (CLI) 4](#_Toc165449717)

[a) Requirement: 4](#_Toc165449718)

[b) Welcome Message: 4](#_Toc165449719)

[c) Character Handling: 5](#_Toc165449720)

[d) Newline Handling: 6](#_Toc165449721)

[e) Delete Handling 14](#_Toc165449722)

[f) Auto Completion 15](#_Toc165449723)

[g) Commands History 18](#_Toc165449724)

[3. Further Development of UART Driver 19](#_Toc165449725)

[a) Baudrate configuration 19](#_Toc165449726)

[b) databits configuration 22](#_Toc165449727)

[c) stopbits configuration 25](#_Toc165449728)

[d) Handshaking Configuration 30](#_Toc165449729)

[4. Get Some Used with Some Common Sensors 32](#_Toc165449730)

[a) Ambient light sensor 32](#_Toc165449731)

[b) Gas sensor 34](#_Toc165449732)

[c) PING 28015 Ultra Sonic Distance sensor 36](#_Toc165449733)

[III. Reflection & Conclusion 39](#_Toc165449734)

[IV. REFERENCES 39](#_Toc165449735)

[V. DEMO 40](#_Toc165449736)

# INTRODUCTION

Embedded systems have a important function in a diversity of areas, from consumer electronics to industrial automation. With the growing complexity and diversity of embedded applications, the demand for skilled engineers proficient in bare metal development of operating systems is ever-increasing. This report summary the journey that has been done to design an operating system from scratch during assignment 2 of EEET2490 – Embedded System: OS and Interfacing, offered in Semester 2024-1.

The primary aim of this report is to provide a comprehensive overview of the tasks involved in the assignment, highlight the learning outcomes, and offer students hands-on experience with bare-metal creation of an operating system in Raspberry Pi 3 or 4. Through the implementation of features, student are able to learn practical skills that transcend theoretical knowledge.

# ADDITIONAL FEATURES FOR BARE METAL OS

## Background

Command Line Interpreter (CLI) is a program that allows users to enter commands then executes those commands to the operating system [1]. By using CLI, the user can have direct access to the operating system. Moreover, CLI also performs the task quickly with cost fewer RAM and CPU usage than GUI [2].

For easier understanding, every **`function`** will be **capitalized** and will put between two backtick (`). Every ***variable*** will be **capitalized** and *italic.*

To further customize our command line, we need to use ANSI escape codes [3]. These codes are used to control text formatting, cursor movements, and color output in terminal emulators [4].

+ ‘\e[1;1H\e[2J’: Clear entire screen and move cursor to top-left corner[5].

+ ‘\x1B[30m’: ANSI escape sequence that will change the text to black[3].

+ ‘\x1B’: This is hexadecimal prefix of Escape [3].

+ ‘[’: Control Sequence Introducer (CSI), indicates the beginning of ANSI escape sequence[3].

+ m: the end of escape sequence [3].

+ \b: Backspace [3].

+ \n: Newline [3].

+ \t: Horizontal tab [3].

Moreover, we also need ASCII table to check if user is typing a specific button that are not alphabetical[6]. As we are creating a new OS, we can’t use commands from any prebuild library of C or C++ such as `**strlen`**, `**strcpy`**, `**strcmp`** of the **string.h** library. Therefore, I’ve created a custom function of them for my code.

**`cus\_strcmp`**: to compare between 2 commands and return 0 if it matches.

A screenshot of a computer

Description automatically generated

Figure 1: cus\_strcmp declaration in kernel.c

**`cus\_strlen`**: to check for command’s size return length of command.

A screen shot of a computer program

Description automatically generated

Figure 2: cus\_strlen declaration in kernel.c

**`cus\_strtok`**: to take a divide a long command into different parts.

A screen shot of a computer program

Description automatically generated

Figure 3: cus\_strtok declaration in kernel.c

**`cus\_strcpy`**: to copy a command to another command.

A screen shot of a computer program

Description automatically generated

Figure 4: cus\_strcpy declaration in kernel.c

**`cus\_parseint`:** used to convert string to integer

A computer screen shot of text

Description automatically generated

Figure 5: cus\_parseint declaration in kernel.c

## Welcome message and Command Line Interpreter (CLI)

### **Requirement:**

At the start of the program, it will print out some basic information like course name, project name, student name, and student ID. User than will have 4 commands to interact with the OS including help, clear, setcolor and showinfo. Help will show all valid commands, you can go to more detail by using Help <commands>. Clear command will clear all commands on screen and put cursor on top-left corner of the screen. Setcolor will change text and background color of the terminal. Showinfo will printout board’s information. The CLI should support features like auto-completion when pressing the Tab key, command history when pressing the ‘\_’ and ‘+’ keys.

### Welcome Message:

Whenever starting the program it will print out basic information of the program following with the NingOS: > waiting for command input.

A black screen with white text

Description automatically generated

Figure 6: Running Attempt of Welcome Messages in Tera Term

**\*\*Code Breakdown:**

A screenshot of a computer screen

Description automatically generated

Figure 7: Welcome Messages Code in kernel.c

**`uart\_puts`** is the function that will print out characters to terminal through Uart.

### Character Handling:

**\*\*Code Breakdown**

The function `**cli`** reads the characters from Uart input by **`uart\_getc`** and save into ***cli\_buffer***. And print it back to the screen through **`uart\_puts`** if the input character is backspace (\b), ‘\_’ (95 in ASCII) or ‘+’ (43 in ASCII) it will not print to the screen. And if the character is different than newline characters (\n), ‘\_’, ‘+’ or backspace, it will save that character into ***cli\_buffer***

***A computer screen shot of text

Description automatically generated***

Figure 8: Character Handling Code in kernel.c

### Newline Handling:

**\*\*Code Breakdown**

**A computer screen with white text

Description automatically generated**

Figure 9: Newline Handling Code in kernel.c

If user enters a newline, it will set value of the ***identicalLock*** flag to zero that will later be used to handle auto completion. print out the commands that have been saved in ***cli\_buffer*** to the screen then it will copy the commands to history that will later be used by showing history function. After that it will put NULL (\0) into ***cli\_buffer*** that has address of ***index***.

A screenshot of a computer screen

Description automatically generated

Figure 10: Errors Declaration in kernel.c

To check for what commands have been input into ***cli\_buffer***, I need to use **`cus\_strtok`** to divide the input into different parts by space characters. By using **`cus\_strcmp`** I can check whether the current token match with any commands. If not **`errors`** function will be executed and print out “Invalid command!”.

+ Case Help:

If a user types in help, it will list out all valid commands. If the user want to know more details about a specific command. For example, clear, it will print out only clear commands.

A black background with white text

Description automatically generated

Figure 11: Running Attempt of help function in Tera Term

A screen shot of a computer

Description automatically generated

Figure 12: Running Attempt 2 of help function in Tera Term

A screenshot of a computer program

Description automatically generated

Figure 13: Run Attempt of Help function on QEMU

**\*\*Code Breakdown**

A screenshot of a computer

Description automatically generated

Figure 14: Help Function Code in kernel.c

If the value of ***token*** is “help”, the system will still scan for another token call ***secondToken*** to check if the next character is NULL or not in case the user wants more details of a specific command. If the user type in another command after “help ” the detail of the specific command will be given. Otherwise, it only print out the syntax of four commands. If the user types in some commands that are not valid, the **`errors`** function will be executed.

+ Case Setcolor:

The user can change text and background color in to one of eights color including black, red, yellow, green, blue, purple, cyan and white.

**\*\*Code Breakdown**

A computer screen shot of text

Description automatically generated

Figure 15: Setcolor Function Code in kernel.c

If the value of ***token*** is “setcolor”, the system will scan for another token call ***secondToken*** to check if the user wants to change background (“-b”) or text (“-t”) color. Then it will scan for another token call ***colors*** if the user wants to change text color, ***backgroundColor*** if user decided to change background color. After getting the token, the system will be executed the **`setColor`** or **`setBackgroundColor`** depending on user choice. **`setColor`** and **`setBackgroundColor`** function both work in the same way is that to check the characters that’s been sent from ***color*** or ***backgroundColor*** by **`cus\_strcmp`.** If it’s match the input (= 0) ***currentColors*** variable will be overwrite and later be used by uart\_puts at the end of the **`cli`** function

A computer screen shot of a code

Description automatically generated

Figure 16:Setcolor Function Code 2 in kernel.c

A screen shot of a computer program

Description automatically generated

Figure 17: Setcolor Function Code 3 in kernel.c

A screen shot of a computer

Description automatically generated

Figure 18: Setcolor Function Code 4 in kernel.c

If the user has already decided to change text color to red, and now they also want to change background color to cyan in one command, the system will still scan for ***thirdToken*** if there’re a space “ ” after ***color*.** The system now will scan for backgroundColor. Same with text color, after typing in background color, user still can change their text color.

+ Case clear:

If user type in “clear”, the system will clear everything currently on the screen and then put the cursor on top-left of the screen.

**\*\*Code Breakdown**

A black background with white text

Description automatically generated

Figure 19: Clear Function Code in kernel.c

If token match with clear, it will run an escape sequence [‘\e[1;1H\e[2J’](#_Background)

+ Case showinfo:

A screenshot of a computer

Description automatically generated

Figure 20: Run Attempt of showinfo on Tera Term

A screenshot of a computer

Description automatically generated

Figure 21: Run Attempt of showinfo on QEMU

The MAC Address got different between two versions.

A screen shot of a computer program

Description automatically generated

Figure 22: Showinfo Code on kernel.c

A screen shot of a computer program

Description automatically generated

Figure 23: Function to print out MAC address and Board Revision

### Delete Handling

**A computer screen shot of a computer

Description automatically generated**

Figure 24: Delete Handling Code in kernel.c

If user uses backspace button, it will set value of identicalLock flag to zero that will later be used to handle auto completion. If the index is bigger than zero, which means that there are some things to delete, it will move cursor to the left, input space and move to the left again to delete the character on the left of the cursor. Then it will minus the current value of ***Index*** by one and set value of ***cli\_buffer[index]*** to NULL to delete the character all of input buffer.

### Auto Completion

Whenever the user tab, a suggested word will be printout and replace the old word.



Figure 25:Input one or two characters.



Figure 26: Auto Completion on QEMU

A red line on a black background

Description automatically generated

Figure 27: Auto completion on Tera Term

If user tabs in, the ***identicalMatchNum*** variable equal to zero will be toggled. The reason for the **`for`** loop conditions is that whenever we tab out it will fill the current word on screen with space so that it will be dividable to seven. For example, if we currently have three characters on screen, it will fill four space characters, if we have eleven it will fill three space characters. Therefore, the loop will run up to “7 – index” equivalent to the number of characters left to fill with space when tab. Therefore, the first 2 **`if`**is to put the cursor back next to the input word. If the ***index*** is less than seven, it will run a **`for`** loop and delete a space by moving the cursor back, input a space and move the cursor back again. If the index is bigger than 7 the loops conditions will be changed and run up to “7-(index – 7)”. The ***identicalLock*** is the variable that will later be used to handle identical cases.

A screenshot of a computer program

Description automatically generated

Figure 28: Auto Completion Code of kernel.c

After that a char pointer is declared with the value of function **`tabHandler`** with the input of ***cli\_buffer*** and ***identicalMatchNum*** variables.

A screenshot of a computer program

Description automatically generated

Figure 29: tabHandler Function of kernel.c

Integer variable ***x***, ***i***, ***numCompletions***, ***lastMatchingIndex***, ***partialLength***, and character pointer that handle more than 1 match command case ***identicalCommands*** will be declared. The first **`for`** loops is to check if input character is similar with given commands. The loops will run up to four times (**COMMANDS\_SIZE = 4**). if two-dimensional array ***commands[i][commandLength]*** and ***cli\_buffer[commandLength]*** different than NULL and current characters of both arrays are match, **`while`** loops will be executed, ***commandLength*** will add one so that the function can continue check the next character of both arrays. If the next character of ***cli\_buffer*** is NULL which means that what user typed in is already checked and all previous characters are matched with a specific command in commands, numCompletions will be added by one to show that one identical command has been found, ***lastMatchingIndex*** will be set to ***i,*** number ofcurrent identical commands.

If only one identical case (***numCompletions*** = 1) and the function does not currently handle more than 1 identical commands (***identicalLock*** == 0), the function will return commands that are matched on previous `**for`** loop.

If more than one identical command is found, data of ***identicalLock*** flag will be set to one, and return the value of ***identicalCommands[currentMatch],*** the value of ***currentMatch*** will always be toggled between zero and one whenever user use tab.



Figure 30: Variable inside tab handling function in kernel.c

So that whenever the flag ***identicalLock*** is one the **`tabHandler`** will always return value of “setcolor” (identicalMatchNum = 0), or “showinfo” (identicalMatchNum = 1). The flag will be reset if user enter a newline or delete the commands.



Figure 31: Variable inside newline and delete function in kernel.c

### Commands History

After type in “\_” or “+” the commands in the past will replace the current on screen.

A red line on a black background

Description automatically generated

Figure 32: Commands History on Tera Term

A black rectangle with white lines

Description automatically generated

Figure 33: Commands History on QEMU

**A black screen with white text

Description automatically generated**

Figure 34: Save Buffer Into History Code in kernel.c

After using some commands, the commands will be saved in a global char pointer name history. ***historyList*** is the counter to show how many history commands that are currently saved in ***history,*** will always plus by one after user enter a newline. And it will update the ***currentHistoryList*** that will later be used to handle command history.

A computer screen shot of text

Description automatically generated

Figure 35: "\_" Handling Code in kernel.c

Whenever user type the underscore sign (“\_”, 95 in ASCII), if the currentHistoryList is smaller or equal to zero, which means that user hasn’t typed in any commands, the function will not be executed. Otherwise, it will minus the currentHistoryList by one and delete any of the current input (if there are one) using **`for`** loop. After that it will copy string from ***history[currentHistoryList]*** to ***cli\_buffer*** by **`cus\_strcpy`** then set index equal to size of ***cli\_buffer,*** and print out the ***cli\_buffer*** to the screen.

A computer screen shot of white text

Description automatically generated

Figure 36: "+" Handling Code in kernel.c

If user type the plus sign (“+”, 43 in ASCII) and the ***currentHistoryList*** is smaller than than ***historyList***, which mean that there are some history commands that have been past through by using underscore (“\_”). The command will be executed. Variable currentHistoryList will be plus by one. Once again, **`for`** loop will be used to delete all current input and then copy string from ***history[currentHistoryList]*** to ***cli\_buffer. Index*** will be set to length of current ***cli\_buffer.*** And `**uart\_puts`** will print out the current value of ***cli\_buffer*** to the screen.

## Further Development of UART Driver

### a) Baudrate configuration

A screenshot of a computer

Description automatically generated

Figure 37: Run Attempt of 57600 Baudrate in Tera Term

A screenshot of a computer

Description automatically generated

Figure 38: Run Attempt of 9600 Baudrate on QEMU

The function is used to change baudrate of the system. However after changing the baudrate, the characters on the screen is unreadable as Tera Term still used the old baudrate to read the system. After changing the suitable baudrate, the code performed perfectly.

**\*Code Breakdown**

A computer screen shot of a program code

Description automatically generated

Figure 39: Baudrate Configuration Code in kernel.c

If token is “baudrate”, the system will ask users to enter their baud rate. I created a **`do-while`** here to get ***baudrateChar*** from uart. The loop will run until the user enter a newline or the index reach its maximum. Because this function runs a different buffer with `**cli`** I need to create for it’s a delete function to handle if the user want to retype the variable. After the loop, we end the buffer with a NULL. I then declared ***baudrate*** to store user input as integer. **`uart\_init`** will be called so that it will re-run everything in **`uart\_init`** again. Then save ***baudrate*** into ***currentBaud*** so that created a current default function for every time **`uart\_init`** is recalled, ensured that the current value won’t change unless we do so. Last **`for`** loop to completely clear the buffer and reset ***baudrateIndex*** to zero.

A computer screen shot of text

Description automatically generated

Figure 40: baurate formula in uart.c

### b) databits configuration

A screenshot of a computer

Description automatically generated

Figure 41: Run attempt of databits configuration in Tera Term

A screenshot of a computer

Description automatically generated

Figure 42: Run Attempt of databit on QEMU

The function is used to change databits of the program. Code will work after adjusting the setup of Tera Term.

**\*Code Breakdown**

**A screen shot of a computer program

Description automatically generated**

Figure 43: Databits configuration Code in kernel.c

Same with baudrate, I still scan for suitable bits and then put it into buffer and then recall **`uart\_int`** and then clear the buffer.

A computer screen shot of a code

Description automatically generated

Figure 44: Databits Configuration in uart.c

I used a switch case to read input and run if case is qualified if case is 5 set LCRH to (0<<5) if case is 6 clear 2 bits at 5 and set (1<<5). Same as the other.

### c) stopbits configuration

A screenshot of a computer screen

Description automatically generated

Figure 45: Run Attempt of stopbit configuration on QEMU

A black screen with white text

Description automatically generated

Figure 46: Bugs before changing stopbits in Tera Term

A screenshot of a computer

Description automatically generated

Figure 47: Work normally after changing the setup of Tera Term

After changing to stopbits 2 the code will still function normally sometime, but sometimes it gets bugs with delete function and if you typing too much or too fast it will bring out some NULL characters. After setting up the Tera Term, stopbits work perfectly.

A screen shot of a computer program

Description automatically generated

Figure 48: Code of stopbit in kernel.c

A computer screen shot of a program

Description automatically generated

Figure 49: Code of stopbit in uart.c

**d) Parity Configuration**

**A screenshot of a computer

Description automatically generated**

Figure 50: Run Attempt of Parity Even on QEMU

**A screenshot of a computer program

Description automatically generated**

Figure 51: Run Attempt of Parity Odd in Tera Term

A screenshot of a computer

Description automatically generated

Figure 52: Run Attemp of Parity Odd after changing setup

A black background with red text

Description automatically generated

Figure 53: Run Attempt of even from none in Tera Term

A black background with red lines

Description automatically generated

Figure 54: After changing setup of Tera Term

A screen shot of a computer program

Description automatically generated

Figure 55: Parity Configuration in uart.c

### d) Handshaking Configuration

In order to check handshaking, we need to unplug wire of GPIO14. And input some things, at this moment, you will see the whole Tera Term screen is freezing, however if you plug back the GPIO14 and insert a newline, all of the characters has been saved to buffer.

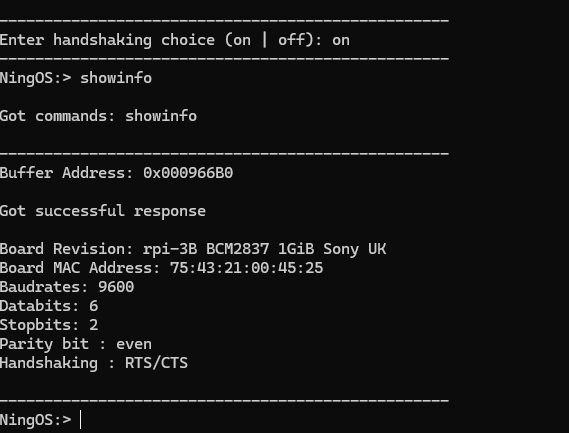


Figure 56:Run Attempt of Handshaking on QEMU

A black background with red lines

Description automatically generated

Figure 57: When unplug the pin

A black background with red lines

Description automatically generated

Figure 58: When replug and type newline

Summary of features implemented in both Task 1 & 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature Group | | Command/ Feature | Implementation | Testing (any issues/limitations) |
| Basic Commands | | help | Complete | No issues |
| clear | Complete | No issues |
| setcolor | Complete | No issues |
| showinfo | Complete | Mac address of Tera Term is on the wrong format |
| CLI enhancement | | OS name in CLI | Complete | No issues |
| Auto-completion in CLI | Complete | No issues |
| Command history in CLI | Complete | No issues |
| UART settings | Baud rate | baudrate | Complete | No issues |
| Data bits | databit | Complete | No issues |
| Stop bits | stopbit | Complete | No issues |
| Parity | parity | complete | No issues |
| Hanshaking | handshaking | complete | I don’t know how to test for it |

## Get Some Used with Some Common Sensors

### a) Ambient light sensor

**\*Sensor Functionality**

Ambient light sensor or ALS is a component that is commonly found in electronics devices such as smartphones, tablets, CCTV, or light system in a smart home. It works by detecting intensity of light in the environment and then adjusting the display brightness of the device accordingly [7]. For example, in a smartphone, ALS can detect whether the surrounding environment is too bright or too dark. Based on that information, ALS can adjust the smartphone screen to the brightness acceptable to human eyes, so that greatly improves the user experience. Ambient light sensor is typically used in controlling the backlight on a display to extend battery life.

**\*Pin Functionality**

A circuit board with wires

Description automatically generated

A close-up of a transistor

Description automatically generated

The ambient light sensor has two pins, controller and emitter. Controller will be connected to 5v pin of a board. emitter will connect through a resistor and then connect to GND. Signal will also be connected via emitter.

**\*Circuit Operation**

A circuit board with wires

Description automatically generated

If the surrounding light is more than 518 it will turn off the led on Pin 2

A circuit board with wires

Description automatically generated

**\*Use of sensor**

These sensors are also used in some types of light source in a smart home so that I can control the house lighting based on environment brightness. Moreover, it can also be used outdoor on street lighting to turn on the light when the sky is getting dark.

**\*Comparing to other devices**

Compared to some other ambient light sensors device on the market, this ambient light sensor look much more simple than the other one. Real-world sensors may offer higher accuracy than just a small 2 pin out device. They can also adjust correctly how strong the screen light should be based on the environment. Not just on and off like our circuit.

### b) Gas sensor

**\*Sensor Functionality**

A gas sensor is a tool for identifying and quantifying the gases present in the environment. The presence and concentration of gases in the surrounding air are then determined by processing and interpreting the output signal.

**\*Pin Functionality**

A circular object with blue lines

Description automatically generated

+ B1, H2, B2: These pins are used to provide power to the gas sensor.

+ A2: This pin is used to transfer data to the board.

+ A1, H1: These pins are used to connect to GND with A1 connected to resistor for safety of components.

**\*Circuit Connection**

A circuit board with wires connected to it

Description automatically generated

Circuit is match with a gas censor and a piezo B1, H2 and B2 of the gas sensor will connect with 5v pin. A1 and H1 connect with GND and A2 connect with pin A0 of the board. The negative side of piezo is connected to GND and the positive size of piezo is connected to pin 8 of the board.

**\*Circuit Operation**

If the gas goes near the gas sensor and the value of gas sensor passes through 235, the piezo will ring.

**\*Use of sensor**

These sensors are crucial for monitoring air quality, ensuring safety in industrial settings, detecting gas leaks, and controlling emissions in automotive and environmental applications.

**\*Comparing to other devices**

Compared to other devices on the market, they can even realize what type of gas is it with a wider ranges and its concentration in the air that you can keep track all of the information through the application on your phone.

### c) PING 28015 Ultra Sonic Distance sensor

**\*Sensor functionality**

This sensor is commonly used in robotics and electronics projects to find the distance between an object with the device by sending ultrasonic waves and then waiting for the waves to bounce back to measure the time it takes.

**\*Pin Functionality**

A blue and white device with two round circles

Description automatically generated

The PING 28015 ultrasonic distance sensor has three pint, GND, 5V and SIG. GND will be connected to GND’s board, 5V will be connected to the power supply of the board and the SIG is the signal output pin.

**\*Circuit Connection**

A blue circuit board with wires

Description automatically generated

The GND pin of the PING 28015 will connect to GND of Uno, 5v pin will connect to 5v power supply of Uno and the Signal Output pin will connect to Pin 7.

**\*Circuit Operation**

**A circuit board with wires and a light

Description automatically generated with medium confidence**

The ultrasonic distance sensor will sent away a sonic wave and if it bounce an object like a ball in this situation, the wave will bounce back to the sensor and give the sensor a times value of its travel. From that data, the sensor can calculate how far the object is.

A diagram of a circuit board

Description automatically generated

**\*Use of sensor**

The PING 28015 ultrasonic distance sensor is often used as object detection on vacuum cleaner robot, it can also be used in parking technology [9]. So that when the car is too near other obstacles, it will ring an alert.

**\*Comparing to another device**

The other ultrasonic distance sensor on the market has a wider range of sonic waves than the PING 28015. The sensor sends back also faster but more accurately than our simulated product.

# Reflection & Conclusion

The project's end results demonstrate a successful enhancement of the foundational operating system intended for embedded applications. Everything from building a dynamic CLI to adjusting the UART driver was done with an eye toward improving the system's performance and responsiveness.

This project has not only strengthened my technical skills and knowledge but also helped me learn how to control a whole project by myself. This is my first time doing an assignment project all on my own, therefore I have faced a lot of challenges in time management. Fortunately, I can finish this project. The toughest part of this whole project is to make the board accept the new configuration from user. Moreover, I also got a hands-on Tinker card, a web-based circuit simulation that might bring a great help to my future study.

# REFERENCES

[1] T. Fisher, “What is a command line interpreter?,” Lifewire, https://www.lifewire.com/what-is-a-command-line-interpreter-2625827 (accessed Apr. 28, 2024).

[2] A. Kumar, “Purpose of the command line interpreter,” Coding Lap, https://codinglap.com/purpose-of-the-command-line-interpreter/#:~:text=The%20benefit%20of%20using%20this,many%20cases%20than%20the%20GUI. (accessed Apr. 28, 2024).

[3] “ANSI escape codes: Enhancing terminal and console applications,” FasterCapital, https://fastercapital.com/content/ANSI-Escape-Codes--Enhancing-Terminal-and-Console-Applications.html (accessed Apr. 28, 2024).

[4] “ANSI escape codes,” Github, https://gist.github.com/fnky/458719343aabd01cfb17a3a4f7296797 (accessed Apr. 27, 2024).

[5] “What does ‘\e’ do? what does ‘\e[1;1h\e[2j’ do?,” Stack Overflow, https://stackoverflow.com/questions/66927511/what-does-e-do-what-does-e11h-e2j-do (accessed Apr. 20, 2024).

[6] “ASCII table,” ASCII Table - ASCII Character Codes, HTML, Octal, Hex, Decimal, https://www.asciitable.com/ (accessed Apr. 20, 2024).

[7] Joseph-Enli, “Ambient light sensor/ALS/working principle/test method/performance parameters/application scope/major manufacturers/industrial structure,” Enlitech, https://enlitech-sensor.com/blog-ambient-light-sensor-01/#:~:text=The%20ambient%20light%20sensor%20can,battery%20life%20of%20electronic%20products. (accessed Apr. 30, 2024).

[8] T. Agarwal, “Ambient light sensor: Types, circuit and applications,” ElProCus, https://www.elprocus.com/ambient-light-sensor-working-and-applications/ (accessed May 1, 2024).

[9] “What is ultrasonic sensor: Working Principle & Applications,” Robocraze, https://robocraze.com/blogs/post/what-is-ultrasonic-sensor#:~:text=Ultrasonic%20sensors%20are%20used%20as,detection%20systems%20and%20factory%20engineering. (accessed May 1, 2024).

# DEMO

<https://youtu.be/rqJpMsj4yP8>