

Department of Electronics & Communication Engineering
SSN College of Engineering

Internship Report : Summer internship report

NAME: BIANCAA.R	
STUDENT ID: 3122223002025	
PROGRAM: B.E ELECTRONICS AND COMMUNICATION	
BATCH: 2022-26	
DETAILS	
Name of organisation: MINDGROVE TECHNOLOGIES.	
Full postal address of organisation: <i>D3-06, IITM RESEARCH PARK, Kanagam Rd, Kanagam, Tharamani, Chennai, Tamil Nadu 600113</i>	
Supervisors name: KAPIL SHYAM .M	
Supervisors job title: SYSTEM SOFTWARE ENGINEER	
Supervisors email: kapil@mindgrovetech.in	
Organisation website: https://www.mindgrovetech.in/	
Start Date: 20.5.2025	End Date: 20.7.25
WORK DONE – KEY POINTS	
<ul style="list-style-type: none">> Setup micropython support for Mindgrove's Secure IoT chip (repl).> Wrote baremetal drivers, compatible to micropython for peripherals in c (GPIO,PWM,UART,SPI,I2C,GPTimer,WDT,RTC) .> Added support for general modules like Numpy,scipy,random,vector,math,ulab,linalg and io.> Added support for time ,with hardware peripheral (clint timer).> Developed a pseudo random logic for random number generation and used it for random module.> Setup a random module from scratch.> Fixed an issue in traps.c ,causing a problem with interrupts identification.> Wrote drivers for getting handlers from both c and python for external interrupts and modified traps.c for the same.> Added support for epoch calculation with rtc ,connected via i2c protocol.> Handled ecall from m-mode exception and several others with OpenOCD ,triggers and GDB while documenting them.> Tried adding support for tflite micro as a module with micropython.> Tried writing the drivers for QSPI.	

INTERNSHIP REPORT

Internship Period: May 2025 – July 2025

Internship Organization: Mindgrove Technologies, Chennai

Designation: System Software Intern

Mode: On-site

Repository: <https://github.com/Mindgrove-Technologies/micropython>

Documentation:

<https://beneficial-death-54e.notion.site/MicroPython-1f9dbd81469780e4927ee078bed18b68?pvs=143>

1. Introduction

I undertook a 2-month internship at **Mindgrove Technologies**, a Chennai-based semiconductor company focused on secure IoT solutions and low-power embedded systems. The internship offered me an exceptional opportunity to work at the intersection of **firmware, hardware abstraction, baremetal programming, debugging, and Python integration**, particularly involving **MicroPython** porting to a custom **RISC-V-based Secure IoT SoC**.

2. Objectives of the Internship

The primary goals set out at the start of the internship were:

- To **enable MicroPython support** for Mindgrove's production-ready secure IoT chip.
- To **develop bare-metal drivers** for various hardware peripherals compatible with the MicroPython runtime.
- To add support for commonly used **Python modules and scientific libraries**.
- To work on **low-level interrupt handling**, system traps, and debug flows using tools like **GDB and OpenOCD**.
- To explore **TensorFlow Lite Micro (TFLM)** integration into MicroPython.
- To provide comprehensive documentation and debugging notes for internal developer use.

3. Work Done

A. MicroPython Porting and REPL Setup

The foundation of my internship was to **bring up the MicroPython interpreter** on the secure IoT SoC designed by Mindgrove. This involved setting up the **REPL (Read-Eval-Print Loop)** to work seamlessly with UART 0 over bare-metal firmware. The chip was emulated in NEXYS Video FPGA Board and through GTKTERM I was passing python commands. The ability to run Python commands in real-time on the chip laid the groundwork for interactive development and module testing.

B. Bare-Metal Driver Development

I was responsible for **developing and integrating bare-metal C drivers**, ensuring they were **MicroPython-compatible**, so that the different peripherals could be controlled easily by **micropython HAL layer itself**:

- **GPIO**: Support for input/output, pin configuration, and interrupts.
- **PWM**: Channel configuration for duty cycle and frequency.
- **UART**: REPL communication and arbitrary serial data exchange.
- **SPI & I2C**: Master-mode support for peripheral communication.
- **GPTimer & WDT**: Timer-based callbacks and watchdog resets.
- **RTC (via I2C)**: Real-time clock initialization, reading epoch time, and setting system time.

Each driver adhered to the low-level HAL APIs and was integrated with the machine module in MicroPython, enabling intuitive use from Python scripts (e.g., `Pin(1, Pin.OUT)` or `UART(1, 9600)`), connected to the baremetal bsp, actually triggering the actions.

C. Module Integration

A significant part of my work included **adding general-purpose Python modules** to the MicroPython build:

- **NumPy, SciPy, ULab**: Enabled lightweight numerical computing through ulab and related C-accelerated functions.
- **Math, Vector, Linalg**: Implemented optimized math functions for embedded applications.
- **Random**: Developed a **custom pseudorandom number generator (PRNG) with LCG algorithm** at the C level and linked it with Python's random module.
- Developed the various random module functions from scratch.
- **Time**: Interfaced hardware **CLINT timer** to provide accurate timekeeping and sleep, ticks, and time() functionalities.

These modules transformed the board into a **computationally capable Python environment**, useful for ML inference, DSP, and real-time logging.

D. Interrupt Handling & Trap Management

I explored the internal **trap handler file (traps.c)** used by the Mindgrove SoC to identify, decode, and redirect interrupt sources:

- Identified and resolved a **bug in the trap handler** that caused misidentification of external interrupts as exceptions.
- Added support to **register interrupt callbacks** from both **C and Python**, linking external interrupt events (GPIO, timer) to Python-level handler functions.
- Deep-dived into **ECALL exception management**, handled via M-mode on the RISC-V core, along with **OpenOCD trigger debugging** and **GDB stepping**.

This taught me about **interrupt vector tables, CSR manipulation, and exception classification**.

E. TFLite Micro Integration (In Progress)

1. I initiated efforts to bring **TensorFlow Lite Micro** inference to the platform by compiling it as a C++ module and linking it into the MicroPython build system. Though complete functionality wasn't achieved within the internship duration, it laid the groundwork for future **edge inference capabilities** on MicroPython.

2. The main reason being the Tensor flow lite micro github repo was completely written in C++, and I would be required to use wrappers to compile them as same object files and link them together and also due to the large magnitude of tflite methods, I was not able to implement it in 2 months period.

F. QSPI Driver Exploration

I began experimenting with the **QSPI (Quad SPI) flash interface**, required for faster data access and potential MicroPython module loading. Though incomplete, this work introduced me to **memory-mapped IO, timing constraints, and multi-line serial data protocol**.

4. Relevance of Academic Courses

Highly Relevant Courses

- **Embedded programming (Hons):** Concepts like memory-mapped IO, timers, and interrupts directly translated into real-world driver development.
- **Advanced microcontrollers:** In this course work I had studied about MSP430 and arm architecture in depth , A lot of concepts in arm architecture was applicable to risc v and I was able to understand concepts like traps, interrupts ,startup files easily.
- **Microcontrollers:** Helped in understanding register-level programming, interrupt prioritization, and SoC architecture.
- **C Programming:** Crucial for writing efficient, low-overhead firmware, clean and concise code.
- **Operating Systems (NPTEL):** System calls, Exception handling, trap vectors, and context switching were central to the trap and ecall handling work.

Less Relevant Courses

- **Analog Circuits:** No direct relevance during the internship.
- **VLSI Design:** Theoretical knowledge, but not required at the driver/software level. As I worked on embedded software side and not in RTL.
- **Control Systems:** Some relevance to timer/PWM-> setup time and hold time calculation, but not explicitly used.

5. Comparison with Academic Learning

Academic learning provided me with a **foundation needed** for applying my knowledge to solve real world problems,This internship showed me the **depth and complexity** of applying those concepts:

Aspect	Academic View	Internship Reality
Interrupts	Theory of priorities and masking	Real trap decoding, GDB debugging, timing
I2C/SPI/UART	Protocol diagrams	Register-level manipulation and debugging
Embedded C	Basic syntax and control structures	Modular, memory-safe driver architecture
MicroPython	Not covered	Interpreter internals, REPL, module binding

6. Lessons Learned

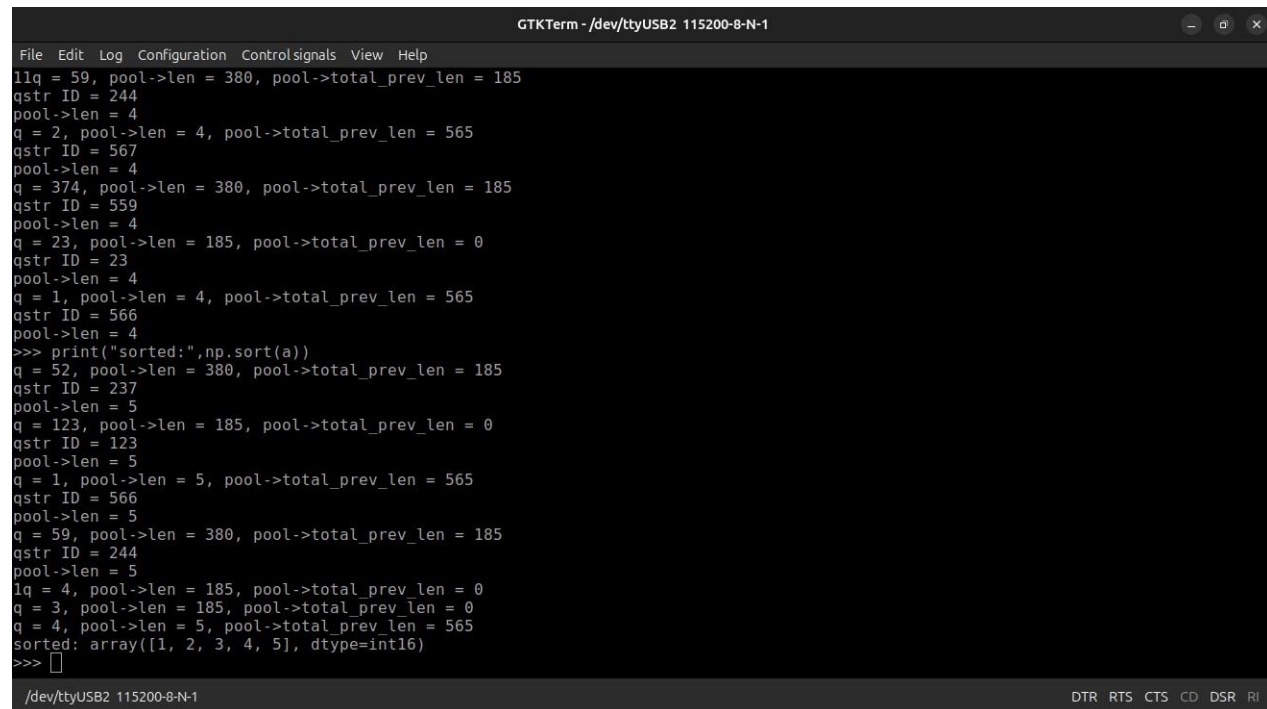
1. **Depth of Firmware Work:** I learned to build things from scratch — not just use APIs, but **write them**, understand the bit level manipulation of the registers of the different peripherals.
2. **Debugging:** Working with GDB, OpenOCD ,triggers, and breakpoints helped me understand the art of **finding invisible bugs**, and correctly identifying the c code with the help of the .elf file , which acts as the root cause of the issue.
3. **Embedded programming with Micropython:** Python, when compiled and tailored, can be extremely powerful on microcontrollers, and a great deal of abstraction can be achieved.
4. **Documentation :** Creating internal documents would help future interns or employees pick up where I left off.

5. **Growth Mindset:** Even when stuck with traps, broken registers, or memory alignment issues, I learned to **embrace the unknown**, and work for it. I had some issues where I had been stuck on for weeks.

7. Conclusion

My internship at Mindgrove Technologies was a **transformative experience**, helping me grow from a student with practical knowledge, but limited to pre-existing microcontrollers like STM32, PIC, avr, MSP430, TM4C123G to an engineer capable of contributing to a **production-level embedded software stack**. It gave me exposure to **low-level system programming, cross-language integration, and real hardware debugging** — the kind of experience that no classroom can replicate. I leave with a deeper sense of confidence, a sharper technical mindset, and a greater respect for the elegance and complexity of embedded systems and firmware development.

OUTPUTS Observed:



```
GTKTerm - /dev/ttyUSB2 115200-8-N-1
File Edit Log Configuration Controlsignals View Help
llq = 59, pool->len = 380, pool->total_prev_len = 185
qstr ID = 244
pool->len = 4
q = 2, pool->len = 4, pool->total_prev_len = 565
qstr ID = 567
pool->len = 4
q = 374, pool->len = 380, pool->total_prev_len = 185
qstr ID = 559
pool->len = 4
q = 23, pool->len = 185, pool->total_prev_len = 0
qstr ID = 23
pool->len = 4
q = 1, pool->len = 4, pool->total_prev_len = 565
qstr ID = 566
pool->len = 4
>>> print("sorted:",np.sort(a))
q = 52, pool->len = 380, pool->total_prev_len = 185
qstr ID = 237
pool->len = 5
q = 123, pool->len = 185, pool->total_prev_len = 0
qstr ID = 123
pool->len = 5
q = 1, pool->len = 5, pool->total_prev_len = 565
qstr ID = 566
pool->len = 5
q = 59, pool->len = 380, pool->total_prev_len = 185
qstr ID = 244
pool->len = 5
llq = 4, pool->len = 185, pool->total_prev_len = 0
q = 3, pool->len = 185, pool->total_prev_len = 0
q = 4, pool->len = 5, pool->total_prev_len = 565
sorted: array([1, 2, 3, 4, 5], dtype=int16)
>>> 
```

Fig 1: Using of numpy module in the REPL to sort an array

```
GTKTerm - /dev/ttyUSB2 115200-8-N-1
File Edit Log Configuration Controlsignals View Help
>>> time.time_ticks_ms()
q = 60, pool->len = 417, pool->total_prev_len = 185
qstr ID = 245
pool->len = 417
q = 371, pool->len = 417, pool->total_prev_len = 185
qstr ID = 556
pool->len = 417
1
>>> time.time_ticks_ms()
q = 60, pool->len = 417, pool->total_prev_len = 185
qstr ID = 245
pool->len = 417
q = 371, pool->len = 417, pool->total_prev_len = 185
qstr ID = 556
pool->len = 417
2
>>> time.time_ticks_ms()
q = 60, pool->len = 417, pool->total_prev_len = 185
qstr ID = 245
pool->len = 417
q = 371, pool->len = 417, pool->total_prev_len = 185
qstr ID = 556
pool->len = 417
2
>>> time.time_ticks_ms()
q = 60, pool->len = 417, pool->total_prev_len = 185
qstr ID = 245
pool->len = 417
q = 371, pool->len = 417, pool->total_prev_len = 185
qstr ID = 556
pool->len = 417
3
>>> □
/dev/ttyUSB2 115200-8-N-1 DTR RTS CTS CD DSR RI
```

Fig 2: Using of time module to setup delays

```
GTKTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Controlsignals View Help
>>> from machine import WDT
q = 230, pool->len = 388, pool->total_prev_len = 185
q = 230, pool->len = 388, pool->total_prev_len = 185
qstr ID = 415
pool->len = 388
q = 230, pool->len = 388, pool->total_prev_len = 185
q = 1, pool->len = 185, pool->total_prev_len = 0
q = 230, pool->len = 388, pool->total_prev_len = 185
q = 43, pool->len = 388, pool->total_prev_len = 185
qstr ID = 228
pool->len = 388
q = 23, pool->len = 185, pool->total_prev_len = 0
qstr ID = 23
pool->len = 388
>>> wdt=WDT(0,500000)
q = 43, pool->len = 388, pool->total_prev_len = 185
qstr ID = 228
pool->len = 1
q = 0, pool->len = 1, pool->total_prev_len = 573
qstr ID = 573
pool->len = 1
q = 23, pool->len = 185, pool->total_prev_len = 0
qstr ID = 23
pool->len = 1
q = 43, pool->len = 388, pool->total_prev_len = 185
qstr ID = 228
pool->len = 1
>>> w.feed()
q = 56, pool->len = 388, pool->total_prev_len = 185
qstr ID = 241
pool->len = 1
q = 379, pool->len = 388, pool->total_prev_len = 185
qstr ID = 564
...
/dev/ttyUSB0 115200-8-N-1 DTR RTS CTS CD DSR RI
```

Fig 3: Using of watch dog timer.

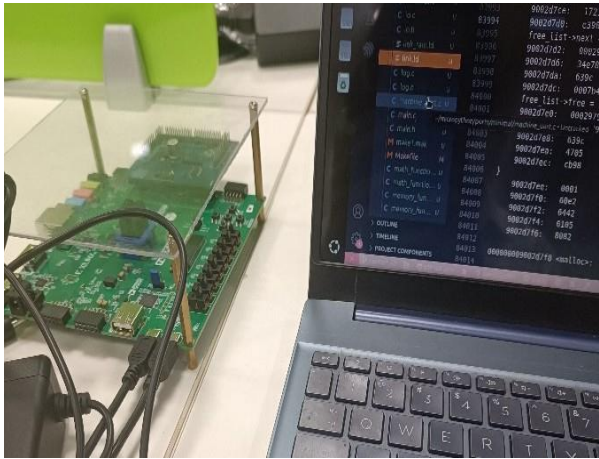
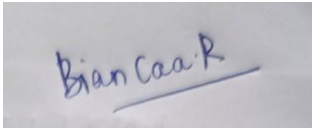



Fig 7: External hardware used for emulation.



Fig 6: Code and experience gained.

AUTHENTICATION:

Student Name: BIANCAA.R	Signature:  20.7.25
-------------------------	---

I certify that this report accurately summarizes the work undertaken by:	
Student Name: BIANCAA.R	
Supervisor Name: KAPIL SHYAM .M	Position: SYSTEM SOFTWARE ENGINEER.
Organisation: MINDGROVE TECHNOLOGIES	
Signature: Kapil Shyam. M 	Date: 06.08.25