GUTHULA SIDDU

Embedded Firmware Engineer

9502598839 guthulasidduvinayakarao@gmail.com Bangalore

Summary

Embedded Firmware Engineer with **1+ year** of experience developing embedded firmware using **Embedded C** on **ARM Cortex-M4 (STM32)** platforms. Hands-on experience in **bare-metal firmware**, **register-level peripheral drivers**, **UART**, **SPI**, **I2C**, **GPIO**, **DMA**, and **interrupt handling**. Experienced in **board bring-up**, firmware validation, hardware-software integration, and debugging using **GDB**, **Logic Analyzer**, **Oscilloscope**, **DMM**, **printk**, and **dmesg**. Currently upskilling in **Embedded Linux** and **Linux Kernel development**, including **Character Device Drivers**, **Linux Kernel Modules**, **IOCTL**, **Device Tree**, **sysfs**, **procfs**, and **user-kernel communication**. Proficient with **GCC**, **Makefiles**, **Git**, and **cross-compilation**.

Professional Experience

Firmware Developer Mar 2025 - Present
VIM SOFT *Bangalore*

- Developed and validated embedded firmware applications using **Embedded C** on **ARM Cortex-M** platforms, supporting peripheral integration and firmware testing.
- Developed register-level drivers for **UART**, **SPI**, **I2C**, **GPIO**, **PWM**, **ADC**, and **Timers** using **Embedded C**, with polling, interrupt-driven, and **DMA**-based communication.
- Performed firmware debugging, root cause analysis, and hardware-software integration using **GDB**, **Logic Analyzer**, **Oscilloscope**, **DMM**, **printk**, and **dmesg**.
- Participated in **board bring-up**, firmware validation, and issue resolution across different hardware configurations.
- Collaborated with hardware and validation teams to analyze and resolve firmware-related issues during peripheral integration and testing.
- Analyzed **hardware schematics**, **datasheets**, **reference manuals**, and **SoC documentation** to support firmware development and peripheral integration.
- Used **Git**, **GCC**, **Makefiles**, and Linux development tools for source control, cross-compilation, and build automation.
- Currently upskilling in **Embedded Linux** and **Linux Kernel development** through hands-on development of **Character Device Drivers**, **Kernel Modules**, **IOCTL**, **sysfs**, **procfs**, and **user-kernel communication**.
- Gained hands-on exposure to **Linux Kernel APIs**, character device registration, file operations, **interrupt handling**, **wait queues**, kernel synchronization, and kernel-level debugging using **printk()** and **dmesg**.
- Used **Python scripting** to automate firmware validation tasks, analyze test data, and support debugging during firmware testing.

Education

RGUKT – IIIT Nuzvid 2021 – 2025
B.Tech – Electronics and Communication Engineering (ECE) *CGPA: 7.23*
RGUKT – IIIT Nuzvid 2019 – 2021
Pre-University Course (PUC) - MPC *CGPA: 8.84*
DR.S.R.K Municipal Corporation High School 2019
SSC *CGPA: 10.0*

Technical Skills

Programming Languages: C, Embedded C, Python (Basic Scripting)

Firmware Development: Bare-Metal Programming, Register-Level Programming, ARM Cortex-M4, STM32, Interrupt Handling, DMA

Microcontrollers/Microprocessors: STM32 (ARM Cortex-M4), ARM Cortex-A72

Communication Protocols: UART, SPI, I2C

Embedded Peripherals: GPIO, PWM, ADC, Timers

Debugging & Analysis: GDB, Logic Analyzer, Oscilloscope, Digital Multimeter (DMM), printk, dmesg, Valgrind

Embedded Linux: Character Device Drivers, Kernel Modules, Linux Kernel APIs, File Operations, IOCTL, Device Tree, sysfs, procfs, Wait Queues, copy_to_user(), copy_from_user(), kmalloc(), kfree(), Cross Compilation, Memory-Mapped I/O

Linux Concepts: Linux Internals, Memory Management, Process Management, IPC Mechanisms, Multithreading, Multiprocessing, RTOS Basics

Tools & Environment: Keil µVision, STM32CubeIDE, GCC, Git, Jira, GitHub, Makefiles, VS Code, Proteus, Arduino IDE

Projects

01. Bare-Metal Firmware Driver Development (UART, I2C, SPI) | *ARM Cortex-M4, Embedded C*

- **UART Driver**
  - Developed a bare-metal **UART Driver** using **Embedded C** and **register-level programming**.
  - Implemented **polling**, **interrupt-driven**, and **DMA-based** communication for efficient data transfer.
  - Configured **baud rate generation**, **parity modes**, **stop bits**, and UART frame formats.
  - Implemented **framing**, **parity**, and **overrun error** detection with recovery mechanisms.
  - Integrated UART interrupts with the **NVIC/vector table** and validated **ISR** functionality through debugging and testing.
- **I2C Driver**
  - Developed an **I2C Master Driver** supporting **Standard (100 kbps)**, **Fast (400 kbps)**, and **Fast+ (1 Mbps)** modes.
  - Implemented **ACK/NACK** handling, **Repeated START** generation, **multi-slave communication**, and **multi-master arbitration**.
  - Added **interrupt-driven** and **DMA-based** non-blocking read/write operations.
  - Validated protocol functionality through communication testing and large-volume data transfers.
- **SPI Driver**
  - Developed an **SPI Master Driver** supporting **full-duplex communication** with configurable **CPOL/CPHA** modes and programmable clock frequencies.
  - Implemented **polling**, **interrupt-driven**, and **DMA-based** data transfers with **multi-slave chip-select** management.
  - Managed **TXE**, **RXNE**, **BSY**, **mode-fault**, and **overrun** conditions for reliable communication.
  - Validated data integrity and timing characteristics through **loopback testing** using a **Logic Analyzer** and **DSO**.

02. Embedded Linux Character Device Driver with IOCTL Interface | *Embedded Linux, Linux Kernel, C*

- Developed a Linux Character Device Driver enabling secure communication between user-space applications and hardware devices through the `/dev` interface.
- Implemented file operations including `open()`, `read()`, `write()`, `unlocked_ioctl()`, and `release()` using Linux kernel APIs.
- Designed and implemented structured IOCTL interfaces for runtime device configuration and control.
- Implemented safe user-kernel communication using `copy_to_user()` and `copy_from_user()` mechanisms.
- Integrated `sysfs` and `procfs` interfaces for runtime monitoring, configuration, and debugging.
- Implemented interrupt handling, wait queues, and kernel synchronization mechanisms to support reliable concurrent device access.
- Performed kernel-level debugging using `printk()`, `dmesg`, and Linux kernel logs to analyze and resolve driver issues.
- Managed character device registration, memory allocation, and resource cleanup during kernel module initialization and removal.