Firmware Advanced Math Library

1. Project Overview

1.1 Introduction

- **Project Name:** `adv-math`
- **Tag**: `ADVM`

1.2 Goal, objectives and scope

- **Goal:** Build a reusable, lightweight, and testable math library for 32-bit firmware (MCUs). This includes basic math operations, algebra, some of advanced math, numerical methods…
- **Output:** `.h/.c` files, test suite, usage examples, documentation.
- **Target Platforms:** STM32, ESP32, Nordic SoCs, or any 32-bit MCU.
- **Non-goals:** No dynamic memory, no OS dependency, no printf in core logic.

1.3 Implementation and Time management

---

2. Software Architecture

2.1. Directory Structure

fw-math/ include/ fw_math.h fw_math_filter.h fw_math_control.h src/ fw_math.c fw_math_filter.c fw_math_control.c test/ test_pid.c test_filter.c examples/ pid_sim.c Makefile README.md

2.2 Library Principles

- **Modular:** Organized by function (math, filter, control, etc.)
- **Portable:** No dynamic allocation, compatible with bare-metal.
- **Scalable:** Support float, fixed-point, and optional bigint later.
- **Testable:** Each module covered by unit tests.
- **MCU-Friendly:** Minimal dependencies, clear API boundaries.

---

4. Milestone Plan

Week	Focus
1	Basic arithmetic (`add`, `mul`, `clamp`, etc.)
2	PID control logic + test
3	Filters (moving average, EMA, etc.)

Week	Focus
4	Usage examples + documentation

---

5. Technical Constraints

Constraint	Detail
Language	C99
Compilers	ARM-GCC, GCC (native)
No dynamic memory	No malloc/calloc/free
Platform compatibility	Buildable for STM32 + native
Binary size	Ideally $<$ 10KB per module

---

6. Tooling

- **Editor:** VS Code
- **Build System:** Makefile or CMake (optional)
- **Unit Test Framework:** Unity / Ceedling or custom minimal
- **Logging:** `fw_log(char*)` hookable by the user
- **Plotting:** Python + Matplotlib
- **CI (optional):** GitHub Actions for native tests

---

7. Code Quality Targets

Metric	Target
Test coverage	90%
Documentation	All public APIs documented (Doxygen-style)
Static analysis	Cppcheck or clang-tidy
Format	clang-format (defined style)
Portability	Zero warnings on both ARM and x86

---

8. Risk Management

Risk	Mitigation
Over-engineering	Iterative releases, working code weekly
Poor reusability	Enforce no globals, pure functions
MCU-specific edge cases	Early native tests, cross-test on STM32 later
Debug difficulty	Pluggable logging, simple trace hooks

---

9. Setup Session Example

In a 4-hour session, aim to: - Create folder structure - Write and test core functions: `fw_math_add`, `fw_math_clamp`, etc. - Add `fw_math_pid.c` with a test - Build a Python script to plot a sample output

---

10. Long-Term Vision

- Fixed-point module (`fw_math_q31`)
- Optional bigint module (`fw_math_bigint`)
- Advanced control (`fw_math_kalman`)
- Auto-generated documentation
- CI/CD pipeline with GitHub Actions
