# RISC-V Audiomark - Coding Challenge

In the coding challenge for this mentorship, you will be implementing a saturating multiply-accumulate function.
You should:

1. Implement a version of the function that is accelerated using RISC-V vector intrinsics (https://drive.google.com/file/d/1RTZi2iOLKzqaX95JCCnzwOm7iCIN3JEq/view?usp=drive_link).
2. Document the design choices and reasoning for the solution you picked
3. Publish your solution on github
4. Preferably: Run your solution on a simulator to check for correctness (e.g. Spike, QEMU, gem5), and obtain some results on the performance of your solution
    a. Alternatively, provide a back of the envelope calculation of expected speed-up.

Note that there is no universally optimal solution. The optimal solution will depend on the input and the machine this is run on.

**1) Implementation Details:**

Implement the following function in C

void q15_axpy_rvv(const int16_t *a, const int16_t *b, int16_t *y, int n, int16_t alpha);

that computes, for all i in [0..n):

y[i]=sat_q15(a[i]+α·b[i])y[i]

**Acceptance criteria**

- **Correctness**: bit-for-bit identical to the scalar reference for all tested inputs (the harness will check).
- **Vector-length agnostic implementation**
- **Buildable** on RV32 or RV64

---

**2) RVV spec & instruction semantics (quick pointers)**

- **RVV Intrinsics v1.0 (ratified)**: naming, typed suffixes, masking & policy variants, vsetvl/vsetvlmax pseudo-intrinsics, and fixed-point rounding (vxrm). https://docs.riscv.org/reference/application-software/vector-c-intrinsics/_attachments/v-intrinsic-spec.pdf
- **RVV specification:** See chapter "V" Standard Extension for Vector Operations in https://docs.riscv.org/reference/isa/_attachments/riscv-unprivileged.pdf

---

### 3) Build & run (example)

Use Clang or GCC as a cross-compiler (see https://github.com/riscv-collab/riscv-gnu-toolchain)

**Example invocation:**

```
riscv32-unknown-elf-gcc
-march=rv32imcbv -mabi=ilp32 \
-O2 src/q15_axpy_challenge.c -o q15_axpy.elf
```

**Run** on your simulator/board; the harness prints verification result and (on RISC-V) rough cycles via rdcycle.

---

### 4) Deliverables

1. Documentation about design choices
2. Link to the published solution on github
3. Possibly measured results

---

### 5) Boilerplate (reference + harness; solution can be inserted)

https://godbolt.org/z/h483Erh7G

Note: Godbolt is a good place to experiment with shorter code snippets, but not a replacement for compiling and simulating in your own environment.