

Techniques for Optimization

2. Identifying work that can be combined

• combine work in time:

• Evaluate in parallel

• Executing SIMD instructions (single instruction, multiple data)

• **Combine work in algorithm:**

element + st: min element st: max element \approx st: min max element



Techniques for Optimisation

2. Identifying work that can be combined

- Combine work in time:
 - Evaluate in parallel
 - Execute using SIMD instructions (single instruction, multiple data)
- Combine work in an algorithm:
 - `std::min_element + std::max_element \approx std::minmax_element`

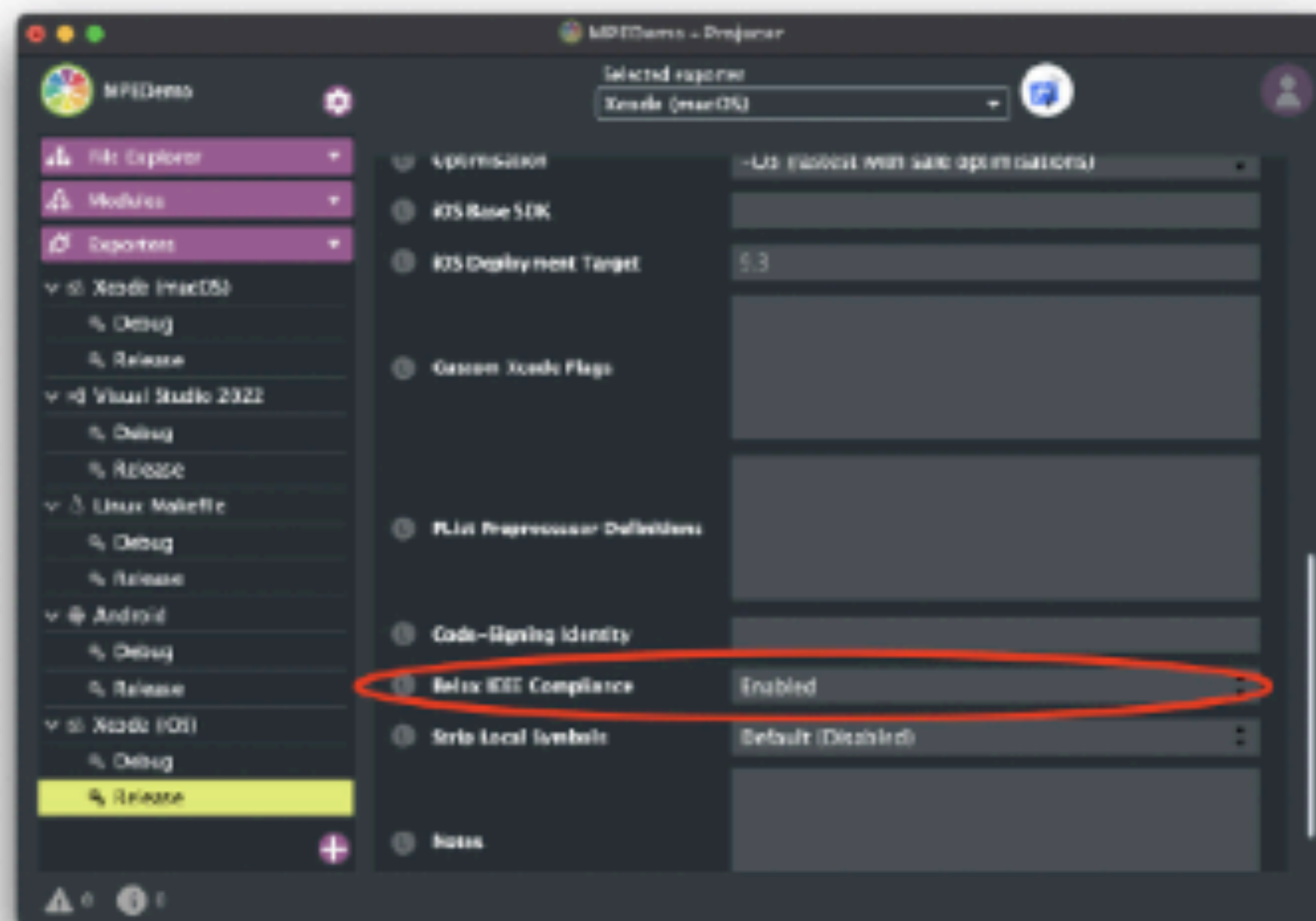
fr810

1 7d

Author of `SIMDRegister` here: I'd just like to add that before embarking on `SIMDRegister`, check that your compiler isn't already auto-vectorizing the tight loops in your code. In my experience, the compiler does a good job auto-vectorizing even moderately complex loops (but see my note at the end of this post).

To ensure that the compiler is allowed to even auto-vectorize your code, be sure that:

1. You are building in release mode (i.e. at least optimisation level `-O3`)
2. You have "Relax IEEE compliance" enabled in the Projucer. This is absolutely crucial, especially on arm/arm64, as SIMD instructions do not have the same denormal/round-to-zero (arm) and/or multiply-accumulate rounding (x86/arm) behaviour as normal IEEE compliant floating point instructions. Hence, the compiler is not allowed to replace your loops with SIMD instructions if it needs to ensure IEEE compliance.



Note if you are compiling with `-Ofast` then "Relax IEEE compliance" will be enabled regardless of the Projucer setting.

To check if your code is being auto-vectorized, you need to look at the assembly. I recommend doing this, even when using `SIMDRegister`, as you will often have unexpected results.