

mmreg

A Rust library and CLI tool for safe, concurrent access to 32-bit memory-mapped IO registers.

Features

- Safe mapping/unmapping of physical memory (e.g., via `/dev/mem`)
- Register abstraction with offset and bitfield helpers
- Grouped register interfaces (e.g., AXI)
- Safe concurrent read/write operations (multi-process)
- Usable as both a library and a command-line tool

Usage

mmreg CLI Usage

Commands

- `read <address>`: Read a 32-bit value from the given physical address.
- `write <address> <value>`: Write a 32-bit value to the given physical address.

Examples

```
mmreg read 0x40000000
mmreg write 0x40000000 0xDEADBEEF
```

Options

- Addresses and values can be specified in hex (with or without `0x`).
- Output is formatted as `0XXXXXXXX` for reads.

Library Usage

See lib.rs for API documentation and examples.

```
use mmreg::{Interface, Register, SubRegister};
use mmreg::{read_register_at, write_register_at};
// Read a 32-bit value from a physical address
let value = read_register_at(0x4000_0000)?;
// Write a 32-bit value to a physical address
write_register_at(0x4000_0000, 0xDEADBEEF)?;
// ...
```

Project Structure

- `src/mapping.rs`: Physical memory mapping
- `src/register.rs`: Single register abstraction
- `src/interface.rs`: Group of registers (interface)
- `src/rw.rs`: Safe concurrent read/write
- `src/lib.rs`: Library API
- `src/main.rs`: Command-line interface

Cross-Compilation Targets

The list of supported cross-compilation targets is maintained in a single file: `targets.txt`.

- **Do not** edit the Makefile or config files to add/remove targets; always update `targets.txt`.
- The Makefile and build automation read from `targets.txt` to ensure consistency.
- Example contents of `targets.txt`:

```
x86_64-unknown-linux-gnu
i686-unknown-linux-gnu
armv7-unknown-linux-gnueabi
aarch64-unknown-linux-gnu
mips-unknown-linux-gnu
powerpc-unknown-linux-gnu
riscv64gc-unknown-linux-gnu
```

To add or remove a target, simply edit `targets.txt` and rerun your build commands.

License

MIT

mmreg Roadmap

Planned Features

- Support for 8, 16, 64, and larger register widths
- Batch/group register operations
- Improved error handling and diagnostics
- More flexible bitfield/subregister API
- Integration tests and example scripts
- Documentation improvements

Future Directions

- Optional async API for high-performance use