⚡ reg-gen

The unified register-map compiler for ASIC designers, verification engineers, and firmware developers.

npm v2.4.0 | MIT-style license | 100% offline

Homepage | Web App | Documentation | Upgrade

reg-gen is an enterprise-grade register management platform that compiles single-source register descriptions into silicon IP RTL, verification environments, software headers, and interactive documentation.

It goes beyond simple code generation — with built-in **Design Rule Checking (DRC/Lint)**, **Semantic Diff & Impact Analysis**, **SoC-level Memory Map Merging**, and support for **6 input/output formats** including SystemRDL and IP-XACT.

Built for ASIC and FPGA teams, it is **100% offline-first** — keeping your proprietary designs entirely local.

6	5	12	0
OUTPUT FORMATS	INPUT FORMATS	LINT RULES	NETWORK CALLS

⚡ Key Features

⚡ 6-Format Compilation

Compiles to RTL, UVM RAL, C Headers, Markdown Docs, IP-XACT, and SystemRDL in a single run.

Bus Protocol Support

Generates synthesizable RTL for APB3, AXI4-Lite, or custom register block architectures.

SoC Memory Map Merging

Ingest 50+ IP specs and merge into a single conflict-free top-level SoC memory map.

Lint / DRC

12 ASIC-focused design rule checks catch common register mistakes before they become silicon bugs.

Diff & Impact Analysis

Compare two spec versions and see exactly which RTL, UVM, C headers, and tests are affected.

5 Input Formats

YAML, JSON, IP-XACT (.xml), SystemRDL (.rdl), and Excel (.xlsx).

Offline Execution

No network calls, telemetry, or cloud-compile steps — your IP never leaves your machine.

Silicon-Ready Output

Synthesizable RTL and complete UVM RAL models generated straight from a single source of truth.

▼ Contents

↓ Installation

Install globally via `npm`:

```
npm install -g reg-gen-cli
```

Or run on-the-fly using `npx`:

```
npx reg-gen-cli --help
```

▶ Quick Start

1 Define your register map

Create `registers.yaml` (e.g., for an SPI Controller):

```
name: spi_ctrl
description: SPI Controller Register Block
baseAddress: 0x40000000
registers:
  - name: spi_status
    description: SPI Status Register
    offset: 0x00
    fields:
      - name: busy
        description: SPI Busy Flag
        bitOffset: 0
        bitWidth: 1
        access: RO
        resetValue: 0
      - name: rx_fifo_empty
        description: RX FIFO Empty
        bitOffset: 1
        bitWidth: 1
        access: RO
        resetValue: 1

  - name: spi_data
    description: SPI Data Register (TX/RX)
    offset: 0x04
    fields:
      - name: data_byte
        description: 8-bit Data payload
        bitOffset: 0
        bitWidth: 8
        access: RW
        resetValue: 0
```

2 Generate all outputs

```
reg-gen generate -i registers.yaml -o ./output -p APB
```

This produces **6 output files** in a single run:

OUTPUT	FILE	DESCRIPTION
SystemVerilog RTL	`<block>_csr.sv`	Synthesizable CSR block with address decoding, read muxes, write-enables. Supports APB or AXI4-Lite
UVM RAL Model	`<block>_ral.sv`	Complete UVM Register Abstraction Layer with uvm_reg_block, uvm_reg, uvm_reg_field
C Header	`<block>_regs.h`	Base address mappings, byte offsets, bitmasks, field shift counts for firmware
Documentation	`<block>_regs.md`	Markdown register manual with address maps, field tables, access policies
IP-XACT 2014	`<block>_ipxact.xml`	IEEE 1685-2014 XML for EDA tool interoperability (Vivado, Platform Designer)
SystemRDL 2.0	`<block>.rdl`	Accellera SystemRDL register description for EDA flows

3 Validate your specification

```
reg-gen validate -i registers.yaml
```

4 Lint your spec for design mistakes

```
reg-gen lint -i registers.yaml
```

```
  Lint Report: spi_ctrl
  ──────────────────────────────

  spi_status
    i INFO  [L006] Read-only field "rx_fifo_empty" has non-zero reset value 0x1.
            Verify this is intentional (e.g., version/ID register).

  Summary: 0 errors, 0 warnings, 1 info
```

5 Diff two spec versions

```
reg-gen diff -a registers_v1.yaml -b registers_v2.yaml
```

```
  Register Diff Report
  ──────────────────────────────
  Before: spi_ctrl (registers_v1.yaml)
  After:  spi_ctrl (registers_v2.yaml)

  Summary: 3 changes · 1 breaking · 2 non-breaking

  spi_status
    ◓ Field Access Changed.busy
      Field "busy" access changed from RO to RW.
      - RO → + RW
      Impact:
        ├── RTL        Read/write enable logic changed (may add or remove write path)
        ├── UVM        Field access policy in RAL changed
        ├── C-HEADER   Field access comments updated
        └── TESTS      Access-type tests need updating
```

6 Merge multiple IPs into a SoC memory map

Create a `soc_manifest.yaml`:

```yaml
name: my_soc
description: "Top-level SoC Memory Map"
subsystems:
  - name: uart0
    baseAddress: 0x40001000
    source: ./ip/uart_regs.yaml
  - name: spi
    baseAddress: 0x40003000
    source: ./ip/spi_regs.json      # cross-format OK
  - name: periph_bus
    baseAddress: 0x50000000
    bridge: AHB-to-APB
    subsystems:
      - name: i2c
        baseAddress: 0x0000          # relative to parent
        source: ./ip/i2c_regs.rdl
```

```
reg-gen merge -m soc_manifest.yaml -o ./soc_output -p APB
```

```
  SoC Memory Map Merge Report
  ──────────────────────────────
  SoC Name:    my_soc
  IP Blocks:   3
  Registers:   8

  Address Map:
  ──────────────────────────────
  0x40001000  uart0           (3 regs, uart_regs.yaml)
  0x40003000  spi             (2 regs, spi_regs.json)
    0x50000000  periph_bus.i2c  (2 regs, i2c_regs.rdl)  [AHB-to-APB]

  ✓ No conflicts detected — clean merge!
  ✓ Merge complete! 3 IP blocks → 8 registers → 6 output files
```

Supported Input Formats

FORMAT	EXTENSION	CLI FLAG	DESCRIPTION
YAML	`.yaml`, `.yml`	`--format yaml`	Human-friendly register specification
JSON	`.json`	`--format json`	Structured JSON register specification
IP-XACT	`.xml`	`--format ipxact`	IEEE 1685-2014 standard XML
SystemRDL	`.rdl`	`--format systemrdl`	Accellera SystemRDL 2.0 register description
Excel	`.xlsx`	`--format excel`	Spreadsheet-based register specification

Format auto-detection: when using the CLI, the input format is automatically detected from the file extension. Use `--format` only to override.

Lint / Design Rule Checking (DRC)

The `lint` command checks your register spec for common ASIC design mistakes that cause silicon bugs, verification headaches, or poor documentation.

ID	RULE	SEVERITY	EDITION	WHAT IT CHECKS
L001	`addr-align`	ERROR	Community	Register offset not word-aligned (mod 4)
L002	`field-overlap`	ERROR	Community	Two fields overlap in bit range
L003	`reserved-gaps`	WARNING	Community	Undefined bit positions — no field covers those bits
L004	`name-convention`	WARNING	Community	Register/field names contain invalid characters
L005	`wo-field`	WARNING	Community	Write-only fields can't be read back — dangerous for debug
L006	`reset-ro-nonzero`	INFO	Community	Read-only field with non-zero reset value
L007	`addr-gap`	WARNING	Pro	Large address gaps between consecutive registers
L008	`field-no-desc`	INFO	Pro	Field missing a description
L009	`reg-no-desc`	INFO	Pro	Register missing a description
L010	`single-bit-no-name`	WARNING	Pro	Single-bit field with a generic name like `bit0`
L011	`reset-mask`	WARNING	Pro	Reset value exceeds the field's bit width
L012	`addr-dense`	INFO	Pro	Register block has >50% address space unused

Lint CLI Usage

```
# Run all community rules (default)
reg-gen lint -i spec.yaml

# Filter by severity — only show warnings and errors
reg-gen lint -i spec.yaml --severity warning

# Strict mode — upgrades all warnings to errors (Pro)
reg-gen lint -i spec.yaml --rules strict

# JSON output for CI/CD integration (Pro)
reg-gen lint -i spec.yaml --json

# Use with any input format
reg-gen lint -i spec.rdl
reg-gen lint -i spec.xml -f ipxact
```

Diff & Impact Analysis

The `diff` command semantically compares two register specs and reports every change with its impact on downstream outputs. This is the feature that prevents broken regressions before tape-out.

CHANGE	SEVERITY	DESCRIPTION
`register-added`	NON-BREAKING	New register added to spec
`register-removed`	BREAKING	Register deleted — RTL, UVM, C headers affected
`register-moved`	BREAKING	Register offset changed — address decoder, firmware affected
`register-desc-changed`	NON-BREAKING	Description text updated
`field-added`	NON-BREAKING	New field added to a register
`field-removed`	BREAKING	Field deleted — RTL write path, UVM, C macros affected
`field-bits-changed`	BREAKING	Bit range changed — masks, shifts, RTL bit selects affected
`field-access-changed`	BREAKING	Access type changed (e.g., RW→RO) — RTL write logic affected
`field-reset-changed`	NON-BREAKING	Reset value changed
`field-desc-changed`	NON-BREAKING	Field description updated

Impact Analysis (Pro)

For each change, the tool maps which output areas are affected:

- **RTL** — Address decoder, read/write mux logic, reset values
- **UVM RAL** — Register model, field access policies, offsets
- **C Header** — #define macros, offset values, bitmasks
- **Documentation** — Register tables, field descriptions
- **IP-XACT** — XML elements, address offsets
- **SystemRDL** — Register types, field properties, instances
- **Tests** — Reset value checks, access-type tests, address-based tests

Diff CLI Usage

```
# Basic diff (community — max 2 registers compared)
reg-gen diff -a spec_v1.yaml -b spec_v2.yaml

# Cross-format diff (any format → any format)
reg-gen diff -a old_spec.json -b new_spec.rdl

# Write Markdown report (Pro)
reg-gen diff -a v1.yaml -b v2.yaml -o diff_report.md

# JSON output for CI/CD (Pro)
reg-gen diff -a v1.yaml -b v2.yaml --json

# Exit code: 0 = no breaking changes, 1 = breaking changes found
# Use in CI/CD pipelines:
reg-gen diff -a main.yaml -b feature.yaml || echo "Breaking changes detected!"
```

SoC Memory Map Merging

Modern SoCs have hundreds of nested IP blocks with complex bus bridges. The `merge` command ingests multiple IP register specs and merges them into a single, conflict-free top-level SoC memory map.

Manifest Format

Define your SoC hierarchy in a YAML manifest file:

```yaml
name: my_soc
description: "Top-level SoC"
busWidth: 32
subsystems:
  # Multi-instance: same IP at different addresses
  - name: uart0
    baseAddress: 0x40001000
    source: ./uart_regs.yaml
  - name: uart1
    baseAddress: 0x40002000
    source: ./uart_regs.yaml

  # Cross-format: JSON, RDL, XML all in one manifest
  - name: spi
    baseAddress: 0x40003000
    source: ./spi_regs.json

  # Nested bus bridge hierarchy
  - name: periph_bus
    baseAddress: 0x50000000
    bridge: AHB-to-APB
    subsystems:
      - name: i2c
        baseAddress: 0x0000      # relative to parent
        source: ./i2c_regs.rdl
      - name: timer
        baseAddress: 0x1000
        source: ./timer_regs.xml
```

Key Features

- **Multi-instance** — Same source spec at different base addresses (e.g., uart0, uart1)
- **Nested hierarchy** — Bus bridges with relative addressing
- **Cross-format** — YAML, JSON, IP-XACT, SystemRDL can be mixed in one manifest
- **Conflict detection** — Address overlaps, name collisions flagged as errors
- **Namespace prefixing** — uart0_CTRL, periph_bus_i2c_STATUS — no collisions
- **Unified outputs** — One set of RTL, UVM, C headers, docs for the entire SoC

Merge CLI Usage

```bash
# Basic merge
reg-gen merge -m soc_manifest.yaml -o ./soc_output -p APB

# With AXI4-Lite and ZIP archive
reg-gen merge -m soc_manifest.yaml -o ./soc_output -p AXI4-Lite --zip
```

⚙ CLI Command Reference

reg-gen generate

Compile register specifications into all output formats.

```
reg-gen generate [options]
```

OPTION	SHORT	DEFAULT	DESCRIPTION
`--input`	`-i`	*(Required)*	Input spec file (.json, .yaml, .yml, .xml, .rdl, .xlsx)
`--output`	`-o`	`./reg_gen_output`	Output directory
`--protocol`	`-p`	`APB`	Bus interface: APB, AXI4-Lite, Custom
`--format`	`-f`	*(Auto)*	Input format: json, yaml, ipxact, systemrdl, excel
`--zip`	`-z`	`false`	Also create a ZIP archive
`--license`	`-l`	*(Auto)*	Path to license key file
`--quiet`	`-q`	`false`	Suppress console output

reg-gen validate

Validate spec syntax and structure without generating files.

```
reg-gen validate [options]
```

OPTION	SHORT	DESCRIPTION
`--input`	`-i`	Input spec file (.json, .yaml, .yml, .xml, .rdl, .xlsx)
`--format`	`-f`	Input format: json, yaml, ipxact, systemrdl, excel

reg-gen lint

Check register spec for design rule violations (DRC).

```
reg-gen lint [options]
```

OPTION	SHORT	DEFAULT	DESCRIPTION
`--input`	`-i`	*(Required)*	Input spec file
`--format`	`-f`	*(Auto)*	Input format: json, yaml, ipxact, systemrdl, excel
`--severity`		`info`	Minimum severity: error, warning, info
`--rules`		`default`	Rule preset: default, strict, relaxed (Pro)
`--json`		`false`	JSON output for CI/CD (Pro)
`--license`	`-l`	*(Auto)*	Path to license key file

reg-gen diff

Compare two register specs and analyze impact of changes.

```
reg-gen diff [options]
```

OPTION	SHORT	DEFAULT	DESCRIPTION
`--before`	`-a`	*(Required)*	Baseline (old) spec file
`--after`	`-b`	*(Required)*	Updated (new) spec file
`--format`	`-f`	*(Auto)*	Input format for both files
`--output`	`-o`		Write Markdown diff report (Pro)
`--json`		`false`	JSON output for CI/CD (Pro)
`--license`	`-l`	*(Auto)*	Path to license key file

reg-gen merge

Merge multiple IP register specs into a unified SoC memory map.

```
reg-gen merge [options]
```

OPTION	SHORT	DEFAULT	DESCRIPTION
`--manifest`	`-m`	*(Required)*	SoC manifest YAML file
`--output`	`-o`	`./reg_gen_output`	Output directory
`--protocol`	`-p`	`APB`	Bus protocol: APB, AXI4-Lite, Custom
`--zip`	`-z`	`false`	Also create a ZIP archive
`--license`	`-l`	*(Auto)*	Path to license key file

Web Application

Use reg-gen directly in your browser at **reggen.vlsiworlds.com/web** — no installation needed.

The web app supports:

- All 5 input formats (JSON, YAML, Excel, IP-XACT, SystemRDL)
- File upload or paste input
- Live preview of all 6 output formats
- AI-powered spec generation from natural language
- ZIP download of all outputs

Licensing & Upgrade

By default, **reg-gen** runs in the **Community Edition** tier:

FEATURE	COMMUNITY (FREE)	PRO (LICENSED)
Generate	Limited to first 2 registers per block, watermark headers	Unlimited registers, no watermarks
Validate	Full	Full
Lint	6 basic rules (L001–L006), max 10 findings	All 12 rules, unlimited findings, JSON output, rule presets
Diff	Basic changes, max 2 registers compared	Full impact analysis, unlimited registers, Markdown/JSON reports
Merge	Max 3 IP blocks per SoC, watermarked output	Unlimited IP blocks, no watermarks

To unlock unlimited generation and Pro features, obtain a license from vlsiworlds.com.

Installing Your License Key

The CLI checks for a license key in this order:

1. **Command-line:** `--license <path>`
2. **Current directory:** `.reg_gen_license` file in your working directory
3. **Home directory:** `~/.reg_gen/license.key`

Setup (Unix / macOS / Linux)

```
mkdir -p ~/.reg_gen
cp /path/to/downloaded/license.key ~/.reg_gen/license.key
```

Setup (Windows PowerShell)

```
New-Item -ItemType Directory -Path "$HOME\.reg_gen" -Force
Copy-Item -Path "C:\path\to\downloaded\license.key" -Destination "$HOME\.reg_gen\license.key"
```

Register Specification Schema

The YAML/JSON register specification follows this structure:

```
name: string              # Block name (required)
description: string       # Block description (optional)
baseAddress: number       # Base address, e.g., 0x40000000 (default: 0)
registers:                # Array of registers (required)
  - name: string          # Register name (required)
    description: string   # Register description (optional)
    offset: number        # Byte offset, word-aligned (required)
    fields:                # Array of fields (required)
      - name: string       # Field name (required)
        description: string   # Field description (optional)
        bitOffset: number     # LSB position, 0-31 (default: 0)
        bitWidth: number      # Width in bits, 1-32 (default: 1)
        access: string        # RW, RO, WO, or W1C (default: RW)
        resetValue: number    # Reset value (default: 0)
```

Access Types

TYPE	DESCRIPTION	RTL BEHAVIOR
`RW`	Read-Write	Software can read and write
`RO`	Read-Only	Software can only read; hardware drives the value
`WO`	Write-Only	Software can only write; reads return 0
`W1C`	Write-1-to-Clear	Software writes 1 to clear; hardware sets the bits

✉ Support & Contact

Homepage	**reggen.vlsiworlds.com**
Web App	**reggen.vlsiworlds.com/web**
Main Site	**vlsiworlds.com**
Support & Sales	**contact@vlsiworlds.com**