AMBA Bus Matrix Configuration Tool

Advanced User Guide

Version 1.0.0

Ultra-Comprehensive Edition
July 2025

AXI4 AXI3 AHB
APB UVM SystemVerilog

About This Guide

Part 1: Introduction - Overview and getting started

Part 2: Architecture - Deep technical details

Part 3: Tutorials - Step-by-step instructions

Part 4: Configuration - Complete parameter reference

Part 5: Integration - RTL and VIP integration

Part 6: Troubleshooting - Solutions to common issues

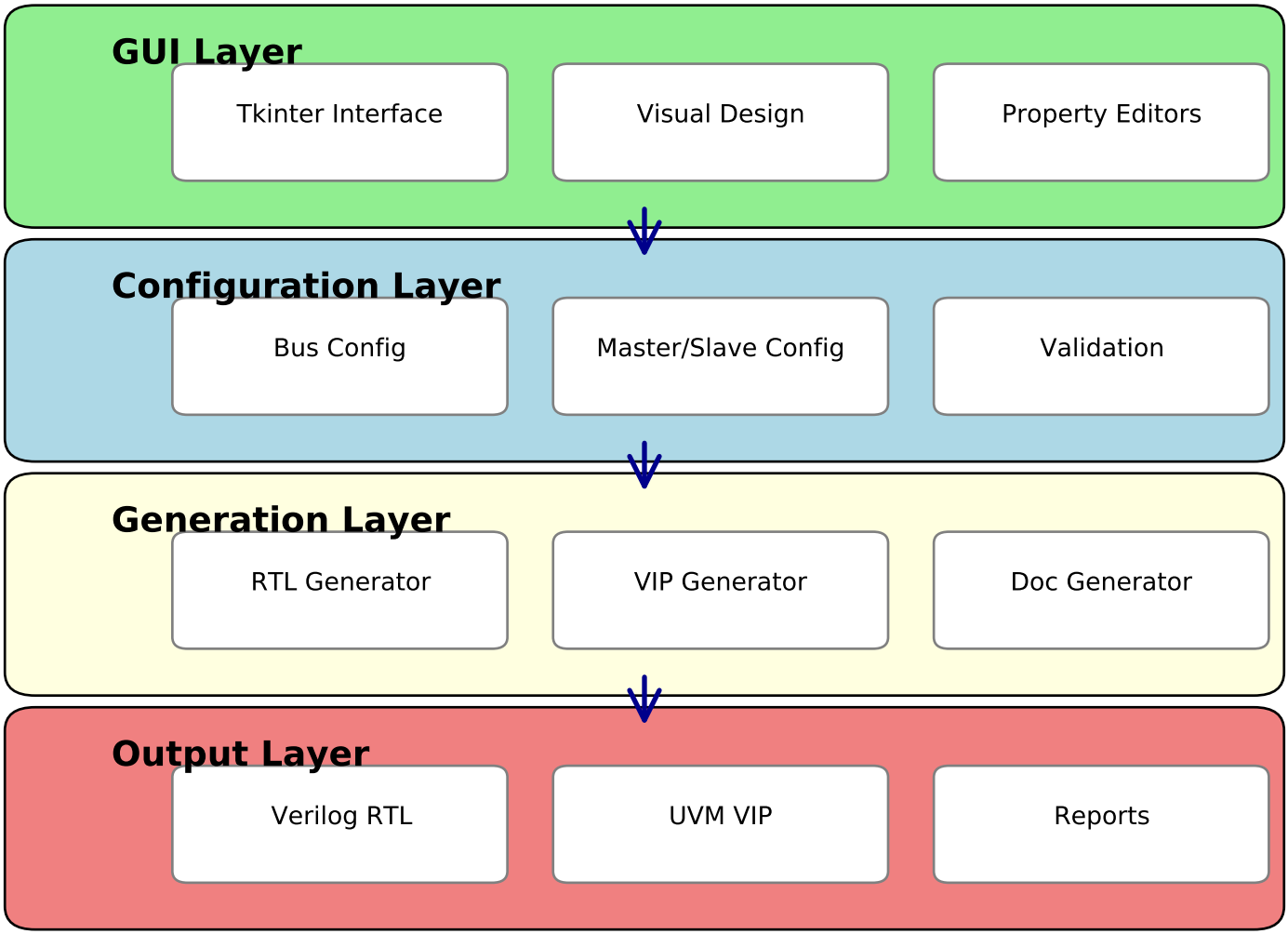
Part 7: Examples - Real-world use cases

Part 8: Appendices - References and glossary

How to Use This Guide:

- Beginners: Start with Part 1 and basic tutorial in Part 3
- Intermediate: Focus on Parts 3-4 for configuration details
- Advanced: Deep dive into Parts 2 and 5 for integration
- Reference: Use Parts 6-8 for troubleshooting and lookups

System Architecture



AMBA Protocol Comparison

	AXI4	AXI3	AHB	AP
Max Burst Length	256	16	Multiple	Single
Outstanding Txn	Yes	Yes	Yes	No
Out-of-Order	Yes	Yes	No	No
Data Width	32-1024	32-1024	32-1024	8-32
Complexity	High	High	Medium	Low
Use Case	High Perf	Legacy	Mid Perf	Peripherals

Protocol Selection Guide:

- AXI4: Choose for new high-performance designs
- AXI3: Use only for legacy compatibility
- AHB: Good for medium complexity systems
- APB: Perfect for low-speed peripherals

AXI4 Signal Reference

Write Address

AWID, AWADDR, AWLEN, AWSIZE, AWBURST

Write Data

WDATA, WSTRB, WLAST

Write Response

BID, BRESP

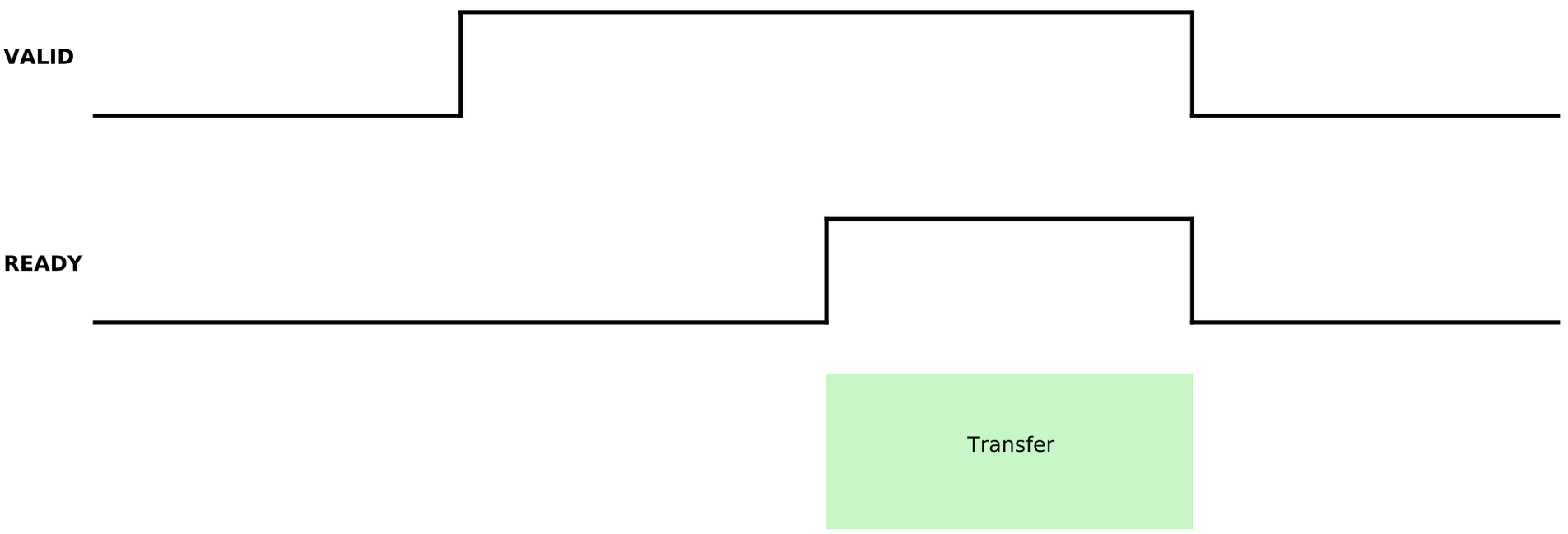
Read Address

ARID, ARADDR, ARLEN, ARSIZE, ARBURST

Read Data

RID, RDATA, RRESP, RLAST

Handshake Protocol



Handshake Rules:

- Transfer occurs when both VALID and READY are HIGH
- VALID must remain stable once asserted
- READY can be asserted before, with, or after VALID

Basic Tutorial: Your First Design

Goal: Create a simple 2×2 AXI4 system

Step 1: Launch the Tool

1. Open terminal
2. Navigate to: cd axi4_vip/gui
3. Run: ./launch_gui.sh
4. Main window appears

Step 2: Add First Master

1. Click "Add Master" button
2. Name: CPU_0
3. ID Width: 4
4. Click OK

Step 3: Add Second Master

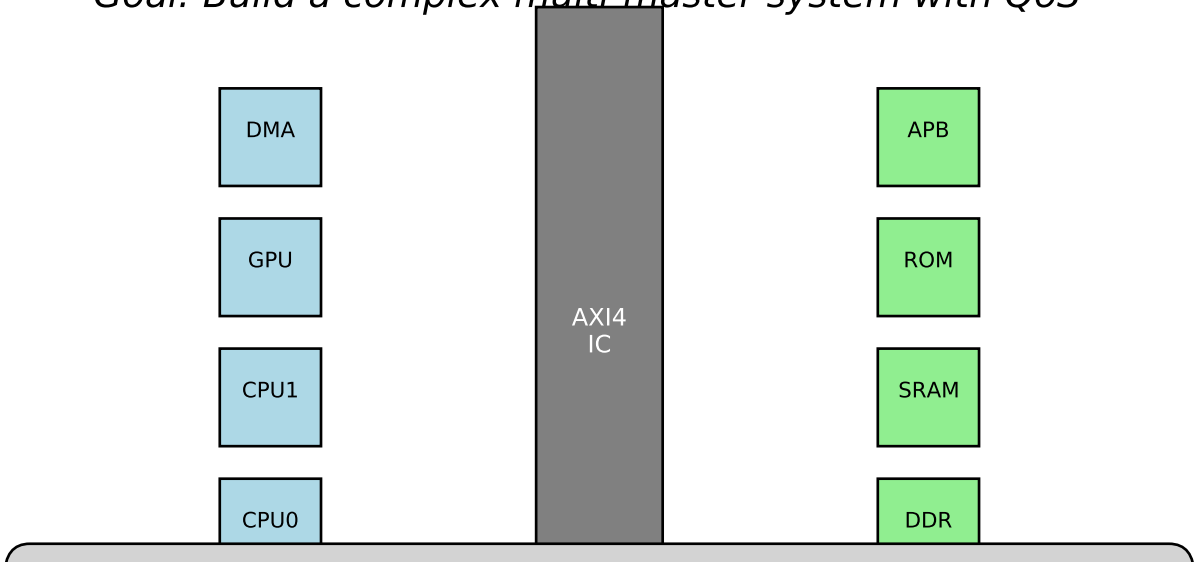
1. Click "Add Master" again
2. Name: DMA_0
3. ID Width: 4
4. Click OK

Step 4: Add Memory Slave

1. Click "Add Slave"
2. Name: DDR_0
3. Base: 0x00000000
4. Size: 1GB

Advanced Tutorial: High-Performance System

Goal: Build a complex multi-master system with QoS



Advanced Configuration:

- Enable QoS for latency-critical masters
- Configure weighted arbitration
- Set up exclusive access monitors
- Define security zones
- Optimize address mapping

Performance Optimization:

- Balance master priorities based on bandwidth needs
- Use appropriate burst sizes for each master
- Minimize crossbar complexity where possible

Expert Tutorial: Custom Extensions

Custom Arbitration Schemes

- Implement deadline-based arbitration
- Add dynamic priority adjustment
- Create custom QoS algorithms

Protocol Bridges

- AXI4 to AXI3 conversion
- Clock domain crossing
- Width conversion

Security Extensions

- Custom security attributes
- Access control lists
- Encryption interfaces

Debug Features

- Transaction trace
- Performance counters
- Protocol checkers

Complete Parameter Reference

Master Parameters

Parameter	Type	Range	Default	Description
name	string	any	-	Unique identifier
id_width	int	1-16	4	Transaction ID width
priority	int	0-15	0	Arbitration priority
qos_enable	bool	0/1	1	QoS support
exclusive	bool	0/1	1	Exclusive access
user_width	int	0-1024	0	User signal width

Slave Parameters

Parameter	Type	Range	Default	Description
name	string	any	-	Unique identifier
base_addr	hex	any	-	Base address
size	int	>0	-	Address range
mem_type	enum	mem/per	mem	Memory type
read_lat	int	>=1	1	Read latency
write_lat	int	>=1	1	Write latency

Design Constraints Guide

Address Constraints

- No overlapping slave address ranges
- Base addresses must be aligned to size
- 4KB boundary rules for AXI
- Power-of-2 sizes recommended

Protocol Constraints

- WRAP burst length: 2, 4, 8, or 16
- Exclusive access max 128 bytes
- Narrow transfers must be aligned
- Unaligned addresses use WSTRB

Performance Constraints

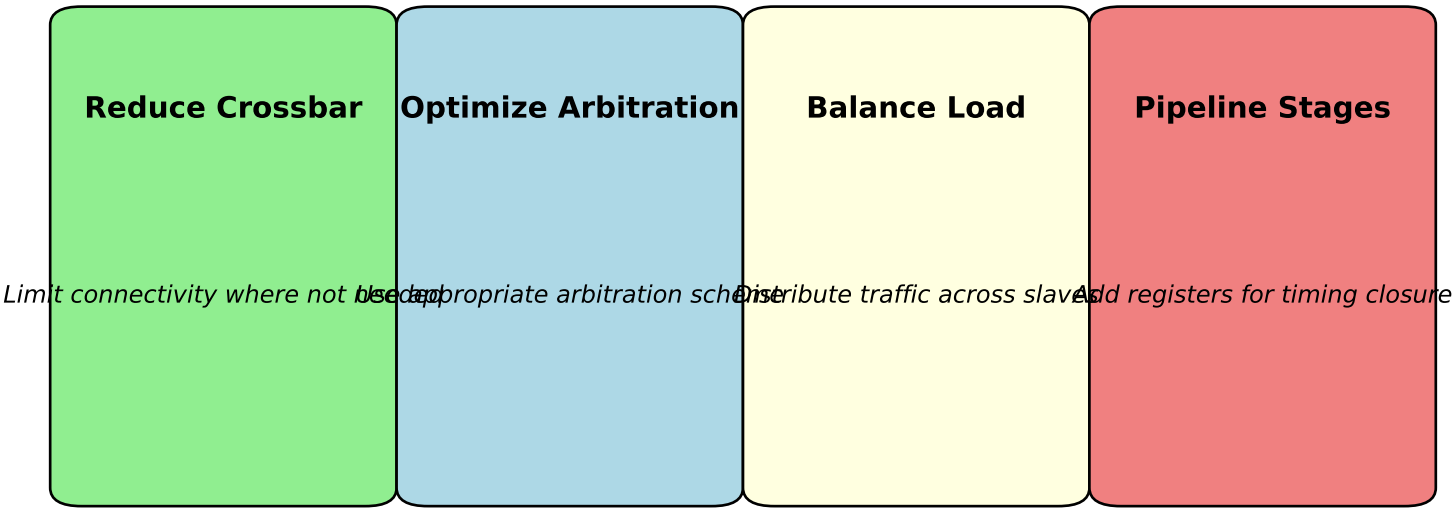
- Max outstanding limited by ID width
- Crossbar complexity grows O(M×N)
- QoS affects arbitration latency
- Wide buses increase area

Security Constraints

- Secure masters → secure slaves only
- Non-secure cannot access secure
- AxPROT must be consistent
- Region settings affect access

Performance Optimization Guide

Optimization Strategies



Key Performance Metrics

$$\text{Bandwidth Utilization} = (\text{Actual} / \text{Theoretical}) \times 100\%$$

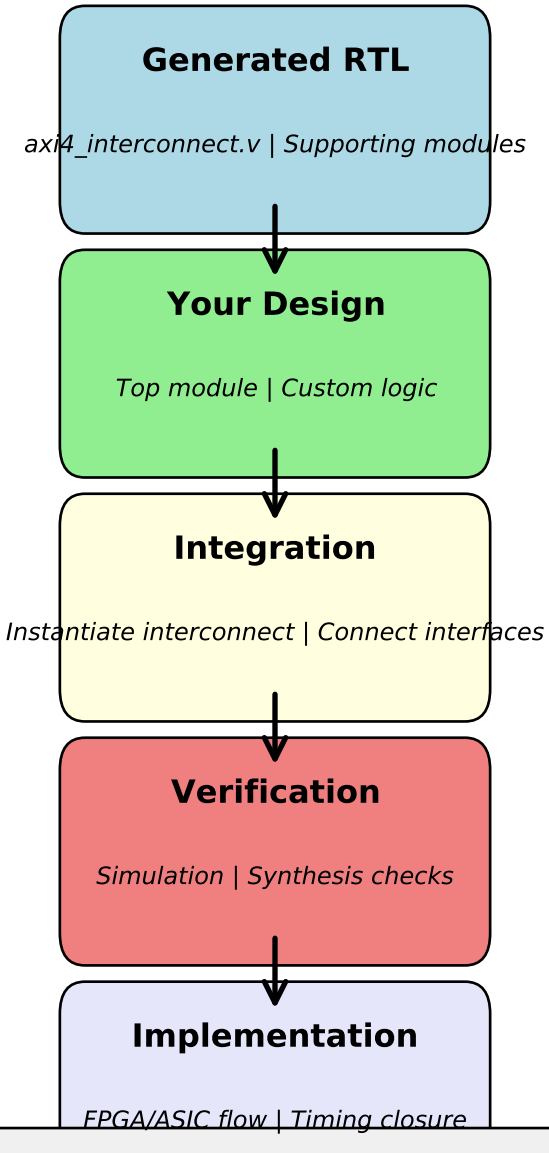
$$\text{Average Latency} = \Sigma(\text{Transaction\_Latency}) / N$$

$$\text{Arbitration Overhead} = \text{Time\_Waiting} / \text{Total\_Time}$$

$$\text{Crossbar Efficiency} = \text{Active\_Paths} / \text{Total\_Paths}$$

RTL Integration Guide

RTL Integration Flow

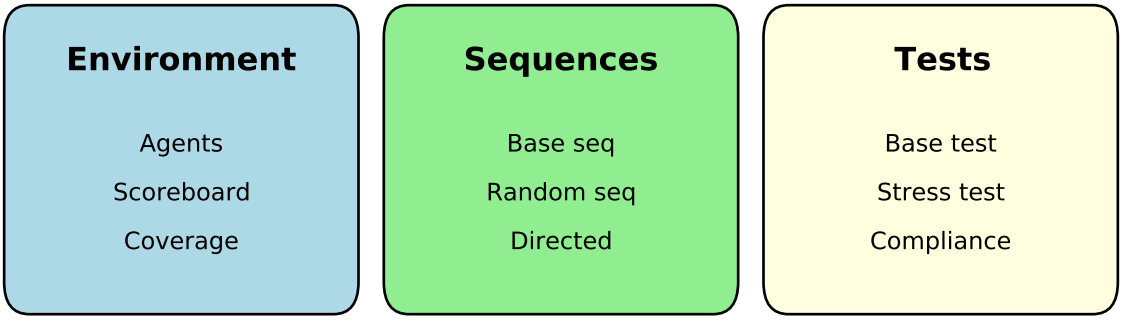


Example Instantiation:

```
axi4_interconnect_m2s3 u_interconnect (
    .clk(clk), .rst_n(rst_n), ...
);
```

VIP Integration Guide

UVM VIP Components



Running VIP Tests:

1. `Compile: vcs -sverilog -ntb_opts uvm`
2. `Run test: ./simv +UVM_TESTNAME=axi4_base_test`
3. `View waves: dve -vpd dump.vpd`
4. `Coverage: urg -dir simv.vdb`

FPGA Implementation Guide

Synthesis

- Run synthesis with generated RTL
- Check for inferred latches
- Review resource utilization

Constraints

- Define clock constraints
- Set I/O timing requirements
- Add false path constraints

Implementation

- Place and route design
- Analyze timing reports
- Optimize critical paths

Verification

- Post-route simulation
- Hardware testing
- Performance validation

Error Reference Guide

Address Overlap Error

Cause: Two slaves have overlapping address ranges

Solution: Adjust slave base addresses or sizes

Port Width Mismatch

Cause: Incompatible data widths between components

Solution: Ensure all widths are compatible or add converters

Invalid Burst Configuration

Cause: WRAP burst with invalid length

Solution: Use only 2, 4, 8, or 16 for WRAP bursts

Security Violation

Cause: Non-secure master accessing secure slave

Solution: Update security settings or connection matrix

Debug Guide

Waveform Analysis

1. Enable transaction recording
2. Look for protocol violations
3. Check handshake timing
4. Verify response codes

Log Analysis

1. Enable UVM verbosity
2. Check error messages
3. Trace transaction flow
4. Monitor coverage gaps

Assertion Debug

1. Review assertion failures
2. Check timing assertions
3. Verify protocol assertions
4. Add custom assertions

Frequently Asked Questions

Q: How many masters/slaves can I have?

A: Practically up to 16 masters and 32 slaves

Q: Can I mix protocols in one design?

A: Yes, use protocol bridges (AXI-to-APB, etc.)

Q: How do I optimize for low latency?

A: Use QoS, minimize arbitration stages, direct paths

Q: What simulators are supported?

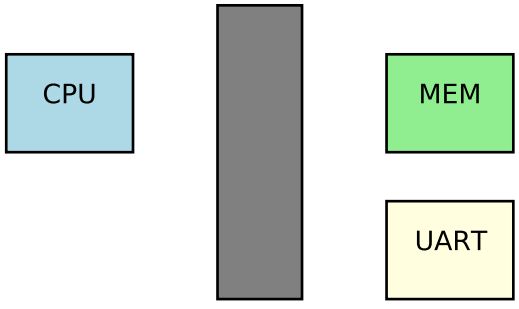
A: VCS, Questa, Xcelium, and Vivado Simulator

Q: Can I modify generated RTL?

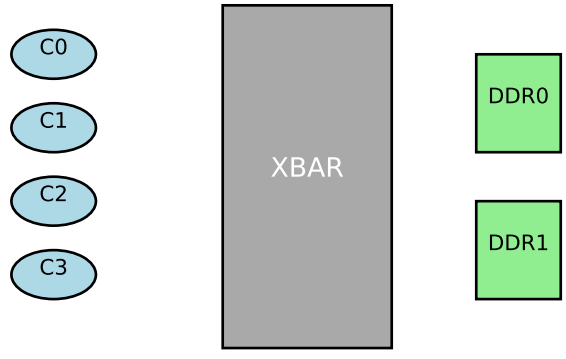
A: Yes, but consider using configuration options first

Example System Designs

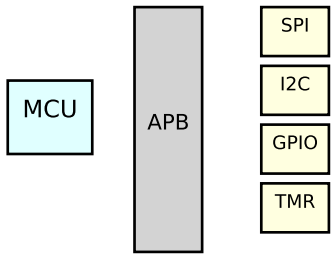
Simple SoC



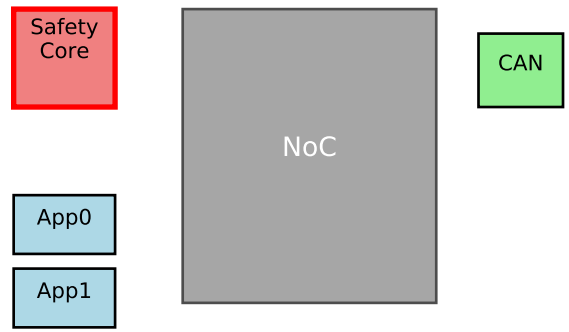
High Performance



IoT System



Automotive



Real-World Use Cases

Mobile SoC

Multi-core CPU with GPU, display, and peripherals

Config: AXI4, QoS enabled, power domains

AI Accelerator

High-bandwidth ML processor with HBM interface

Config: 1024-bit data, multiple outstanding

Automotive ECU

Safety-critical system with redundancy

Config: Lockstep cores, ECC, security zones

Network Processor

Packet processing with multiple interfaces

Config: Low latency, QoS, traffic shaping

Python API Reference

```
class BusConfig:
    add_master(name, **kwargs)
    add_slave(name, base, size, **kwargs)
    validate() -> bool
    save(filename)
```

```
class AXIVerilogGenerator:
    __init__(config)
    generate() -> list[str]
    write_files(output_dir)
    get_file_list() -> dict
```

```
class VIPGenerator:
    __init__(config)
    generate_env()
    generate_tests()
    create_scripts(simulator)
```

Command Line Reference

Launch Commands

```
./launch_gui.sh                # Default launch
```

```
./launch_gui.sh --config file # Load config
```

```
python3 src/bus_matrix_gui.py # Direct launch
```

Generation Commands

```
python3 generate_rtl.py --config config.json
```

```
python3 generate_vip.py --config config.json
```

```
make -C ../.. MST=2 SLV=3    # Makefile
```

Simulation Commands

```
cd vip_output/sim && ./run.sh
```

```
vsim -do run.do              # Questa
```

```
vcs -f files.f && ./simv     # VCS
```

Glossary

AMBA: Advanced Microcontroller Bus Architecture

AXI: Advanced eXtensible Interface

APB: Advanced Peripheral Bus

AHB: Advanced High-performance Bus

QoS: Quality of Service

VIP: Verification Intellectual Property

UVM: Universal Verification Methodology

RTL: Register Transfer Level

DUT: Design Under Test

BFM: Bus Functional Model

PCWM: Port Connection Width Mismatch

ID: Transaction Identifier

WSTRB: Write Strobe signals

AxPROT: Protection type (Access control)

AxLOCK: Lock type (Exclusive access)