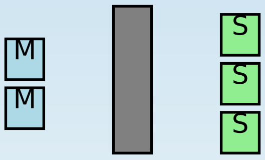
AMBA Bus Matrix Configuration Tool

Step-by-Step User Guide



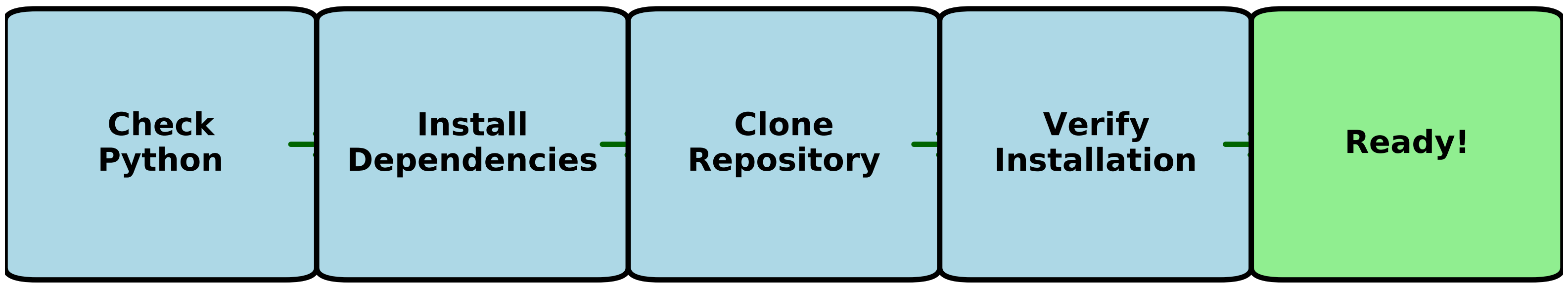
Version 1.0.0
July 2025

 Ultra-Think Edition

Table of Contents

Step 1: Installation and Setup

Installation Flow



Detailed Installation Steps:

```
1. Check Python version (requires 3.6+):
   $ python3 --version
```

```
2. Install pip if not available:
   $ sudo apt-get install python3-pip
```

```
3. Clone the repository:
   $ git clone <repository_url>
   $ cd gen_amba_2025/axi4_vip/gui
```

```
4. Install dependencies:
   $ pip3 install -r requirements.txt
```

```
5. Verify installation:
   $ python3 -c 'import tkinter; print("OK")'
```

⚠ Common Issues:

• No tkinter: sudo apt-get install python3-tk

• Permission denied: use 'pip3 install --user'

• Python too old: upgrade to Python 3.6+

Step 2: First Launch

Launch Methods

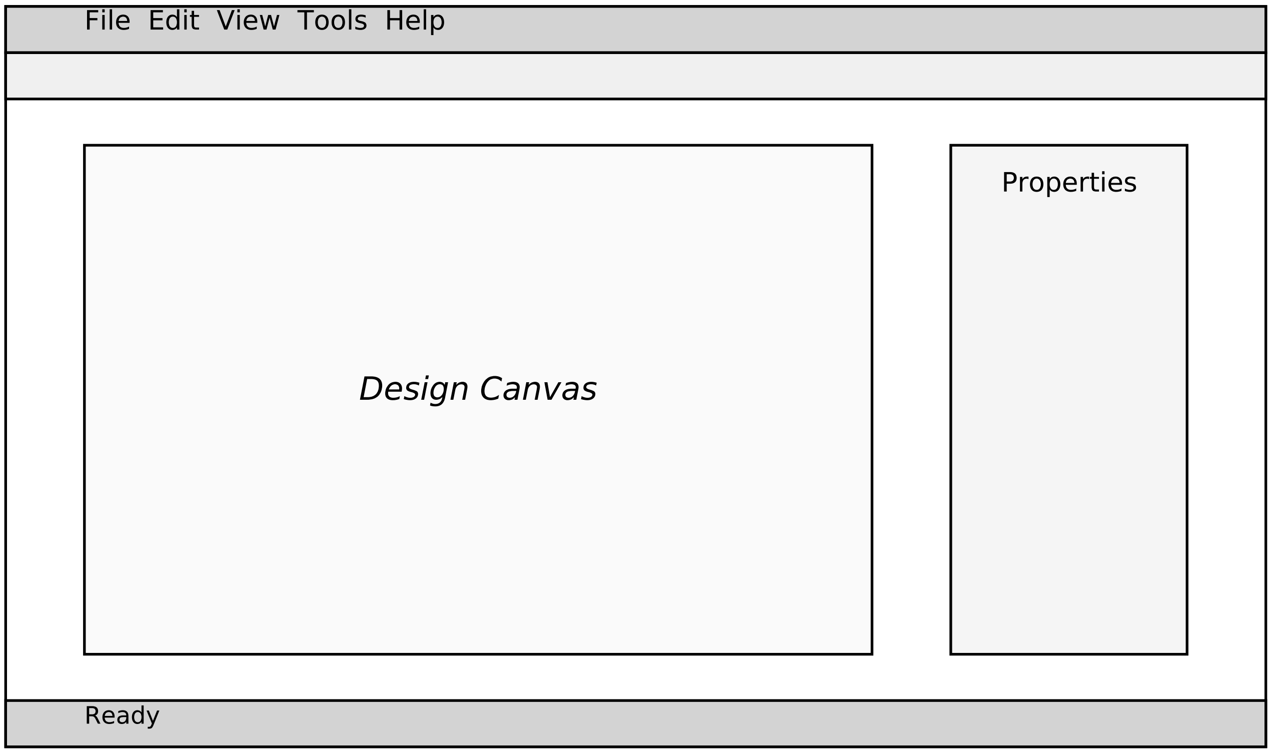
Method 1: Shell Script

```
./launch_gui.sh
```

Method 2: Python

```
python3 src/bus_matrix_gui.py
```

Main Window Layout



Step 3: Creating Your First Design

1. Click 'Add Master'

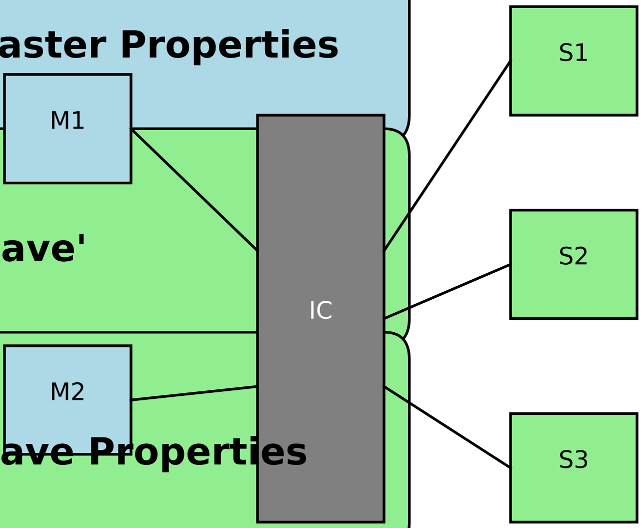
2. Configure Master Properties

3. Click 'Add Slave'

4. Configure Slave Properties

5. Connect Master to Slave

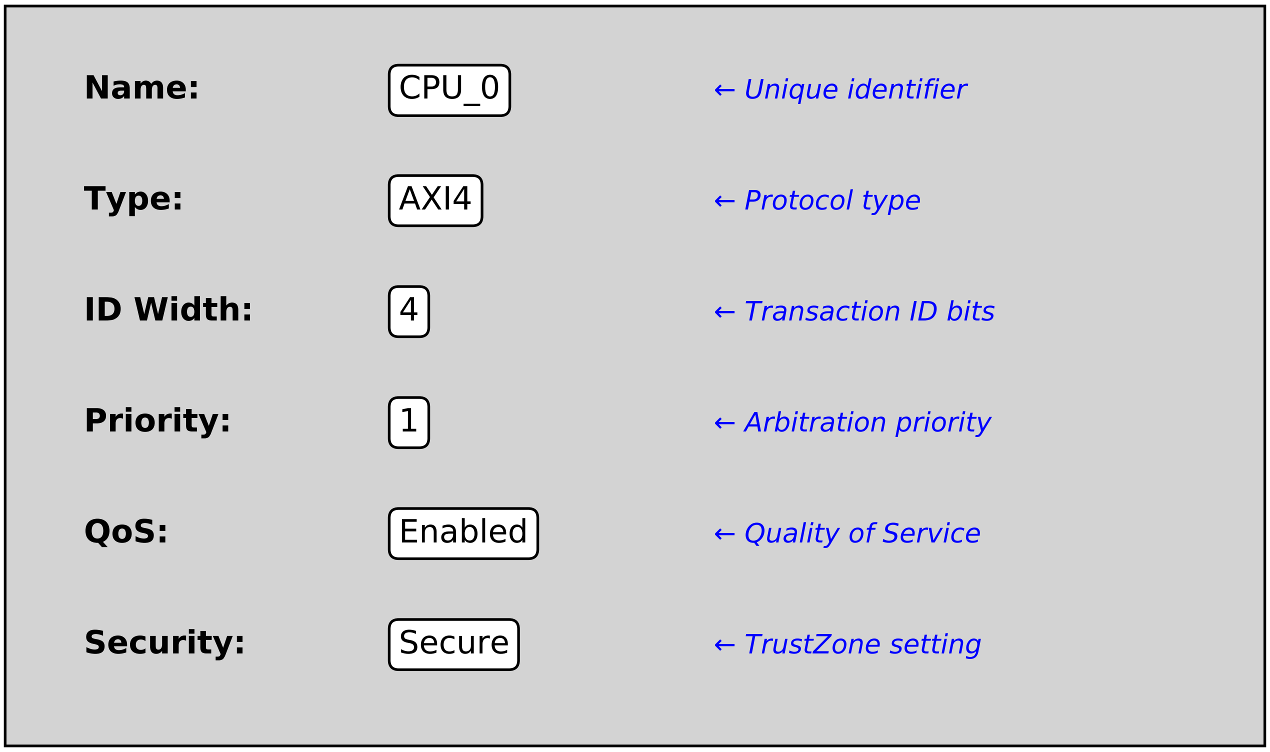
6. Validate Design



□ Tip: Start with a simple 2×2 system to learn the basics

Step 4: Configuring Masters

Master Configuration Panel



Name:	CPU_0	*← Unique identifier*
Type:	AXI4	*← Protocol type*
ID Width:	4	*← Transaction ID bits*
Priority:	1	*← Arbitration priority*
QoS:	Enabled	*← Quality of Service*
Security:	Secure	*← TrustZone setting*

Master Configuration Best Practices

✓ Use descriptive names (CPU_0, DMA_1, GPU_0)

✓ Set appropriate ID width (4-8 bits typical)

✓ Higher priority = higher arbitration preference

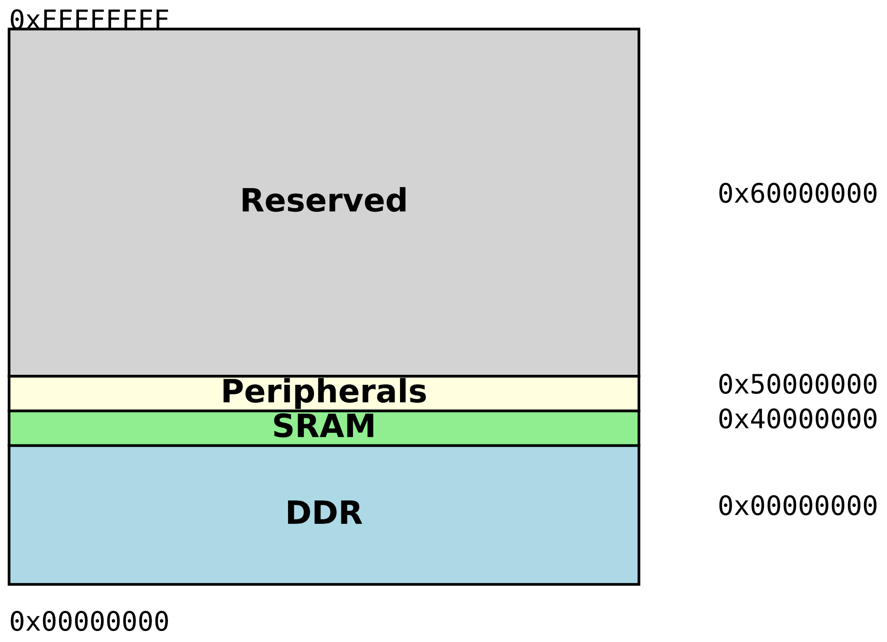
✓ Enable QoS for latency-sensitive masters

✓ Match security settings to system requirements

Step 5: Configuring Slaves

Address Map Configuration

Address Space



Slave Configuration Tips

□ Align base addresses to size boundaries

□ Use power-of-2 sizes when possible

□ Set appropriate access permissions

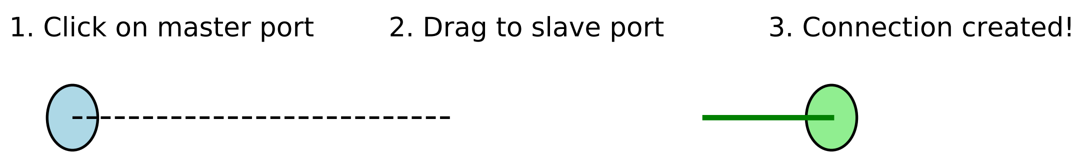
⚡ Configure realistic latency values

□ Avoid address overlaps

Step 6: Making Connections

Connection Methods

Method 1: Drag and Drop



Method 2: Connection Matrix

M/S	S0	S1	S2
M0	✓		✓
M1	✓	✓	
M2		✓	✓

Connection Rules:

- Each master can connect to multiple slaves
- Each slave can be accessed by multiple masters

Step 7: Design Validation

Validation Checklist

✓	**Address Overlap**	*No overlapping slave addresses*
✓	**Connection Check**	*All masters have valid connections*
✓	**ID Width**	*Sufficient ID bits for all masters*
✓	**Data Width**	*Compatible data widths*
✗	**Security**	*Security settings are consistent*
✓	**Performance**	*No bandwidth bottlenecks*

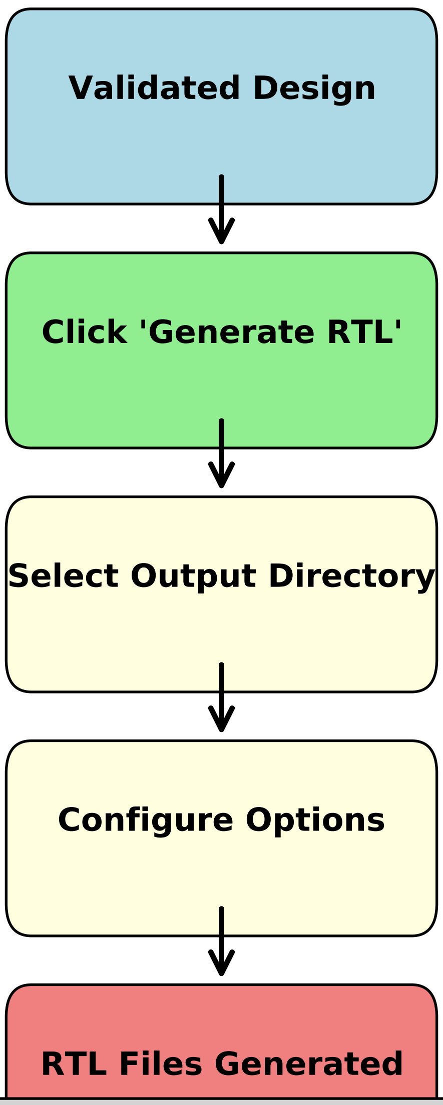
Resolving Validation Errors

□ Security settings are inconsistent

Solution: Ensure secure masters only connect to secure slaves

Step 8: RTL Generation

RTL Generation Flow



Generated Files:

- axi4_interconnect.v - Top module
- axi4_decoder.v - Address decoder
- axi4_arbiter.v - Arbitration
- tb_axi4.v - Testbench

Step 9: VIP Generation

VIP Directory Structure

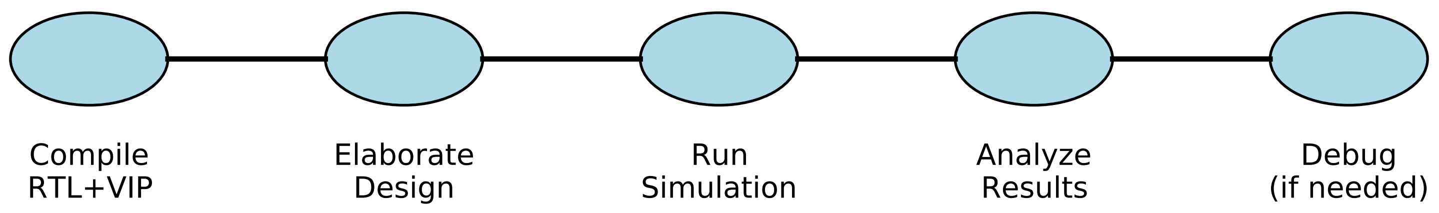
```
vip_output/
├── env/
│   ├── axi4_env.sv
│   └── axi4_scoreboard.sv
├── sequences/
│   ├── axi4_base_seq.sv
│   └── axi4_rand_seq.sv
├── tests/
│   └── axi4_base_test.sv
└── sim/
    └── run.sh
```

To run VIP simulations:

```
1. cd vip_output/sim
2. ./run.sh +UVM_TESTNAME=axi4_base_test
```

Step 10: Running Simulations

Simulation Flow



Simulator Commands

VCS:

```
vcs -sverilog -ntb_opts uvm -f files.f
./simv +UVM_TESTNAME=test
```

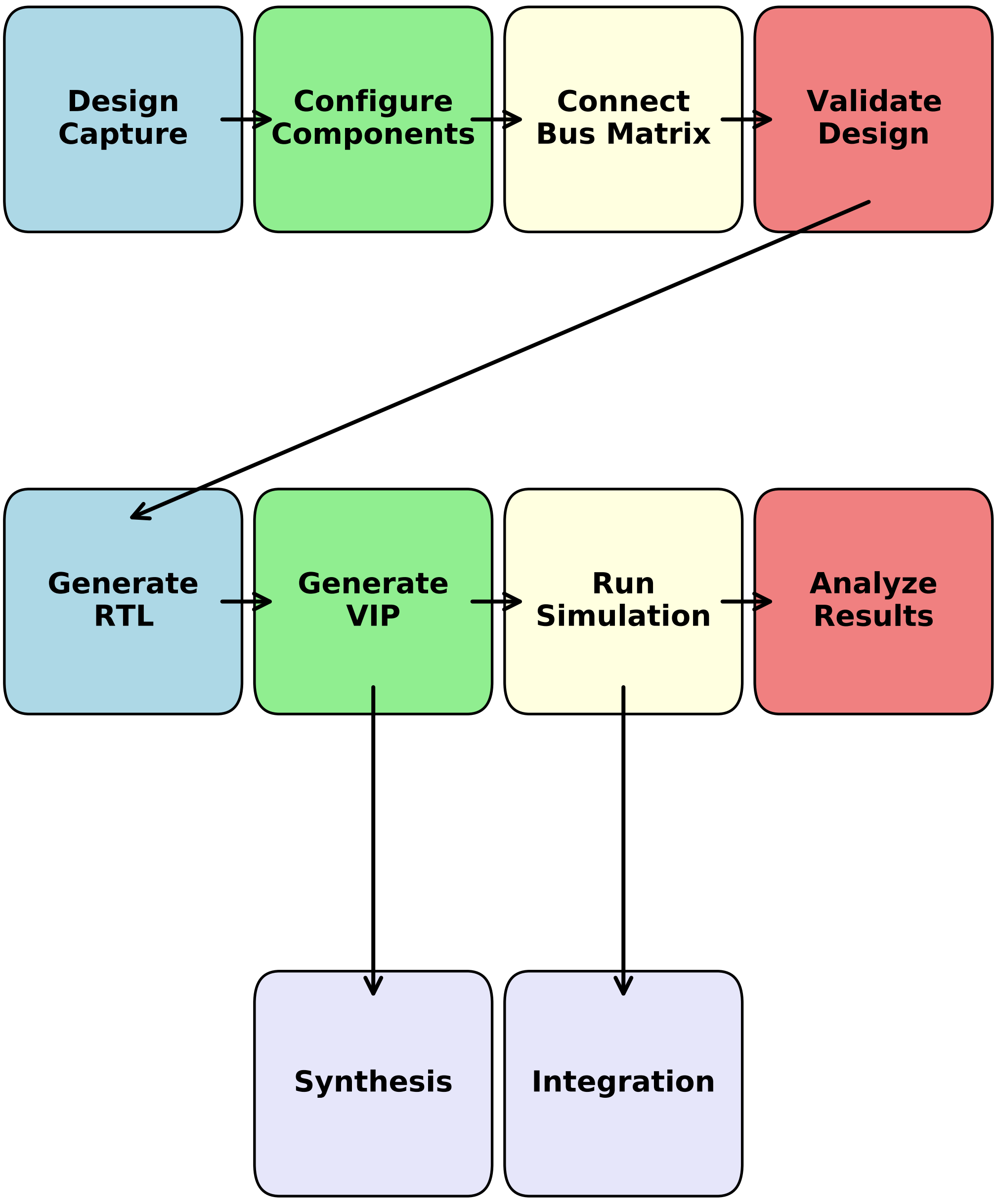
Questa:

```
vlog -sv -f files.f
vsim -c -do "run -all"
```

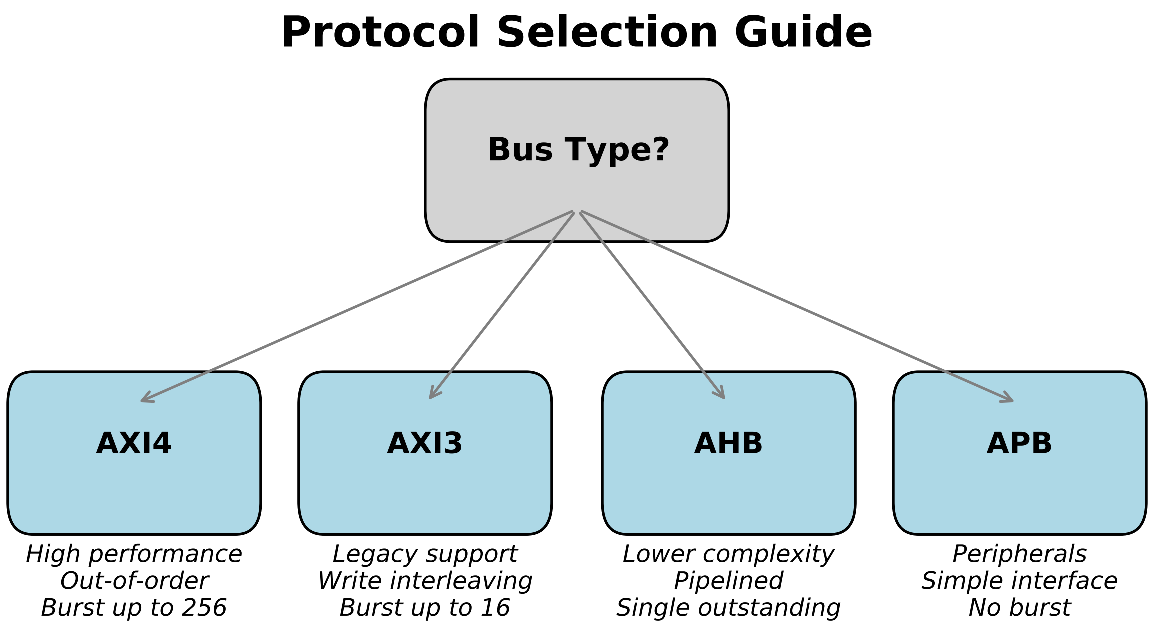
Xcelium:

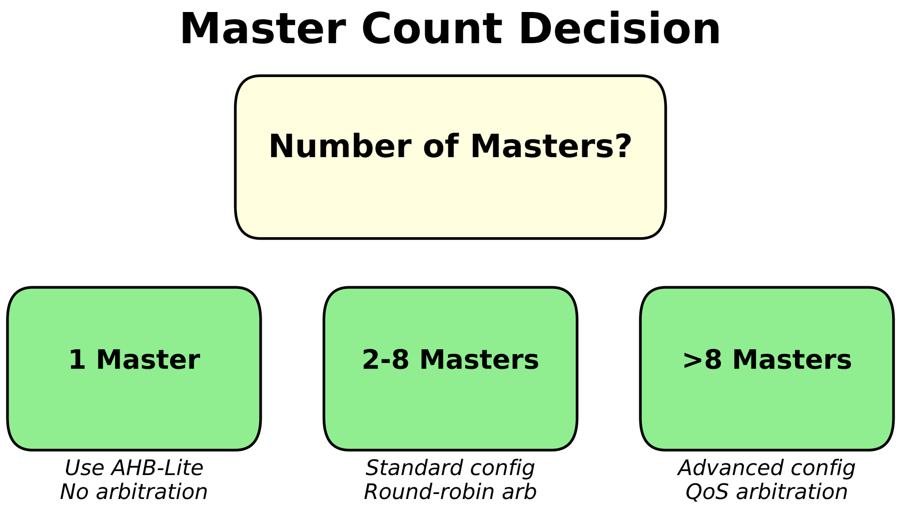
```
xrun -sv -uvm -f files.f +UVM_TESTNAME=test
```

Complete Workflow Diagram



Design Decision Trees





Best Practices

Design Planning

- Start with clear requirements
- Document address map early
- Plan for future expansion

Configuration

- Use meaningful component names
- Set realistic latency values
- Enable only needed features

Validation

- Always validate before generation
- Check address overlaps carefully
- Verify security settings

Performance

- Balance master priorities
- Consider QoS requirements
- Minimize connection complexity

Debug

- Use descriptive signal names
- Enable assertions in RTL
- Generate comprehensive testbench

Quick Reference Card

Keyboard Shortcuts

`Ctrl+N`	New design
`Ctrl+O`	Open file
`Ctrl+S`	Save file
`Ctrl+G`	Generate RTL
`Ctrl+V`	Validate
`Delete`	Remove item

Common Parameters

Data Width:

32, 64, 128, 256

Addr Width:

32, 64

ID Width:

4-16 typical

Burst Len:

1-256 (AXI4)

QoS:

0-15

Important Files

Config:	`*.json`
RTL:	`output/rtl/*.v`
VIP:	`vip_output/`
Logs:	`logs/*.log`

Command Reference

```
./launch_gui.sh
```

```
python3 src/bus_matrix_gui.py
```

```
make -C ../.. cleanup
```

```
pytest tests/
```