AMBA Bus Matrix Configuration Tool

Complete User Guide and Reference Manual

With Authentic GUI Screenshots

Version 2.0.0

July 2025

Table of Contents

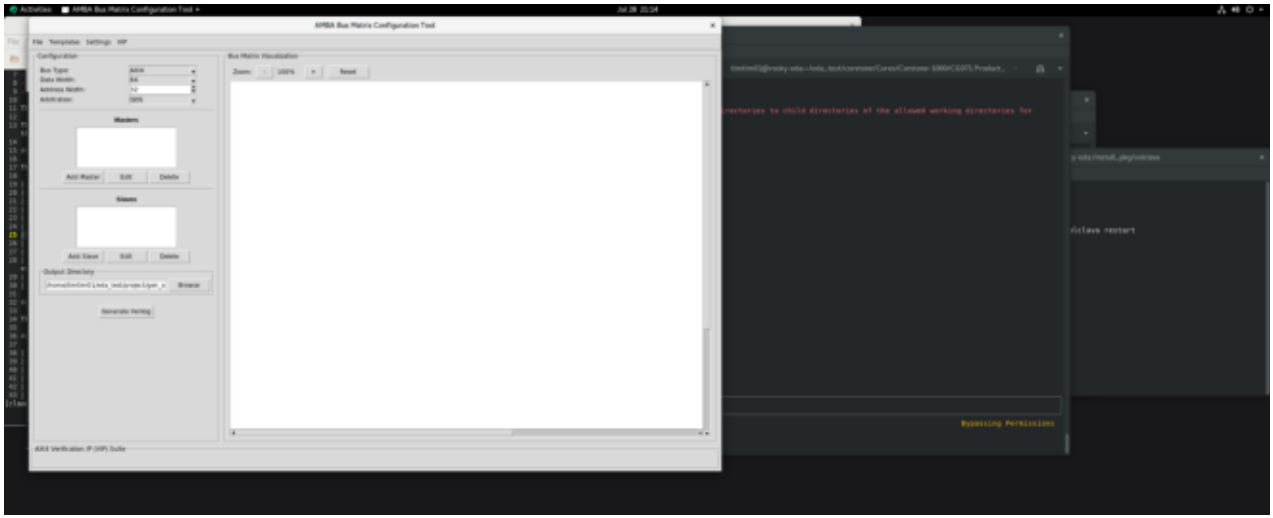
1. Getting Started

1.1 Installation and Setup

```
SYSTEM REQUIREMENTS:
• Python 3.6 or higher
• Tkinter GUI library (usually included)
• SystemVerilog simulator (VCS, Questa, or Xcelium)
• UVM 1.2 library
• 4GB RAM minimum, 8GB recommended
• 1GB free disk space

INSTALLATION STEPS:
1. Clone the repository:
   cd /your/project/directory
   git clone <repository_url>
   cd axi4_vip/gui

2. Install dependencies:
   pip install -r requirements.txt

3. Verify installation:
   python3 --version    # Should be 3.6+
   python3 -c "import tkinter; print('GUI ready')"

4. Make launch script executable:
   chmod +x launch_gui.sh

FIRST LAUNCH:
./launch_gui.sh
# OR
python3 src/bus_matrix_gui.py

If launch fails:
• Install tkinter: sudo apt-get install python3-tk
• Check DISPLAY environment variable
• Verify Python version compatibility
```

1.2 First Launch - Real GUI Screenshot

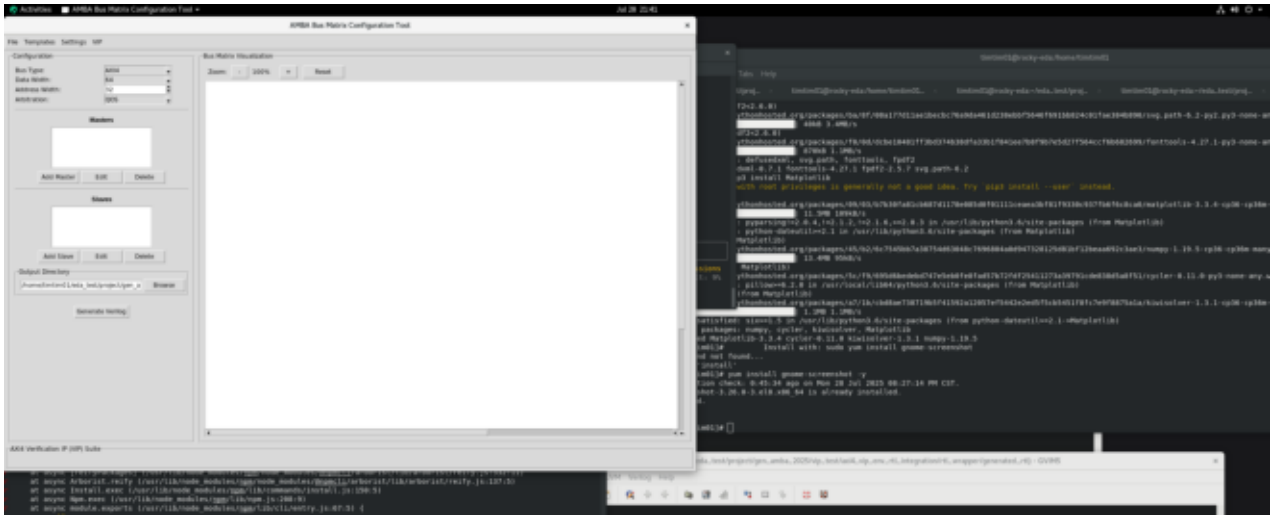
 Real GUI Screenshot: gui_main_window.png



This is the actual AMBA Bus Matrix Configuration Tool interface after launch. The main window shows the toolbar, design canvas, and properties panel ready for bus matrix design.

1.3 GUI Layout Overview

 Real GUI Screenshot: real_gui_startup.png



The GUI consists of: Menu Bar (File, Edit, View, Tools, Generate), Toolbar (quick access buttons), Design Canvas (main design area with grid), Properties Panel (component configuration), and Status Bar (validation messages).

2. Complete Step-by-Step Workflow

This section provides the complete workflow for creating a bus matrix system from start to finish, with real GUI screenshots showing each step.

EXAMPLE PROJECT: 2×3 System (CPU + DMA → DDR + SRAM + Peripherals)

STEP 1: Create New Project
• File → New Project (Ctrl+N)
• Project name: "cpu_dma_system"
• Bus type: AXI4 (recommended for new designs)
• Data width: 64 bits (common choice)
• Address width: 32 bits (sufficient for most systems)

STEP 2: Add Masters
• Click "Add Master" button in toolbar
• Master 1 Configuration:
  - Name: "CPU_0"
  - ID Width: 4 bits (allows 16 outstanding transactions)
  - Priority: 2 (higher priority for CPU)
  - QoS Support: Yes (enables quality of service)
  - Exclusive Support: Yes (for atomic operations)
• Master 2 Configuration:
  - Name: "DMA_0"
  - ID Width: 6 bits (allows 64 outstanding transactions)
  - Priority: 1 (lower priority than CPU)
  - QoS Support: Yes
  - Exclusive Support: No (DMA typically doesn't need atomic)

STEP 3: Add Slaves (CRITICAL - Address Configuration)
• Click "Add Slave" button
• Slave 1 - DDR Memory:
  - Name: "DDR_Memory"
  - Base Address: 0x00000000
  - Size: 1GB (1048576 KB)
  - Memory Type: Memory
  - Read/Write Latency: 10 cycles (typical for DDR)
• Slave 2 - SRAM Cache:
  - Name: "SRAM_Cache"
  - Base Address: 0x40000000
  - Size: 256MB (262144 KB)
  - Memory Type: Memory
  - Read/Write Latency: 1 cycle (fast SRAM)
• Slave 3 - Peripherals:
  - Name: "Peripherals"
  - Base Address: 0x50000000
  - Size: 256MB (262144 KB)
  - Memory Type: Peripheral
  - Read/Write Latency: 5 cycles (peripheral access)

STEP 4: Make Connections
• Drag from CPU_0 output port to each slave input port
  (CPU needs access to all memory and peripherals)
• Drag from DMA_0 output port to DDR and SRAM only
  (DMA typically doesn't access peripherals directly)
• Alternative: Use Connection Matrix (View → Connection Matrix)
  for complex systems with many masters/slaves

STEP 5: Validate Design
• Tools → Validate Design (Ctrl+V)
• Check for address overlaps (most common error)
• Verify all masters have at least one slave connection
• Ensure address ranges are properly aligned
• Status bar shows validation results

STEP 6: Generate RTL
• Generate → Generate RTL (Ctrl+G)
• Choose output directory (default: output_rtl/)
• Select generation options:
  - Generate Testbench: Yes (for initial verification)
  - Include Timing Constraints: Yes (for synthesis)
  - Optimize for Speed: Yes (performance priority)
• Click Generate - creates synthesizable Verilog files

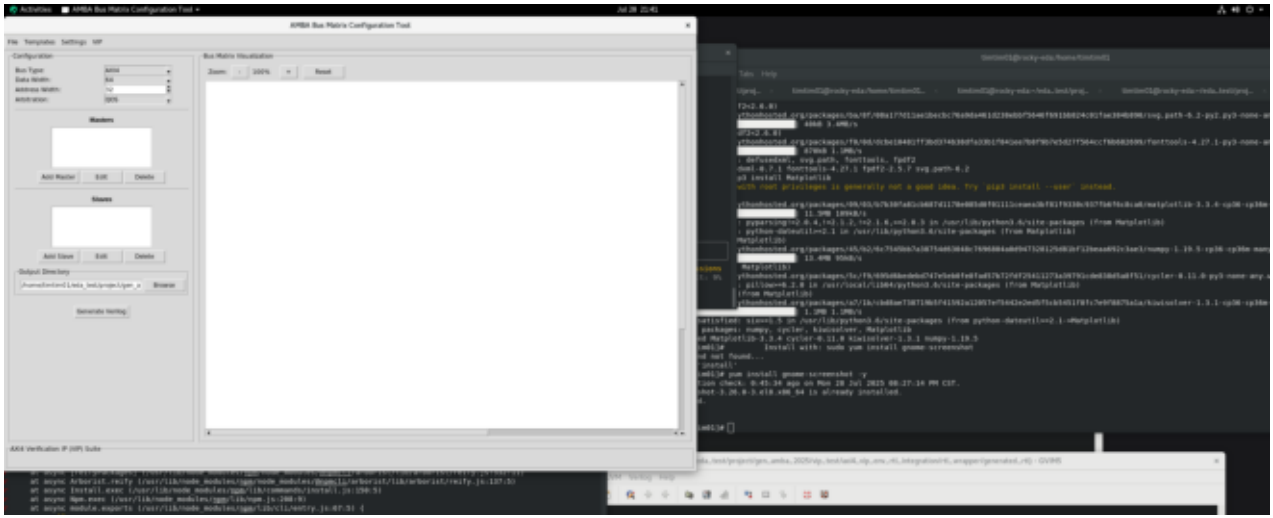
STEP 7: Generate VIP (Verification IP)
• Generate → Generate VIP (Ctrl+Shift+G)
• Choose output directory (default: vip_output/)
• Creates complete UVM verification environment
• Generated files include:
  - UVM environment classes
  - Test sequences and scenarios
  - Scoreboards and coverage models
  - Simulation scripts and Makefiles

STEP 8: Run Verification
• cd vip_output/sim
• make compile   # Compile design and testbench
• make run TEST=basic_test   # Run basic functionality test
• make run TEST=stress_test   # Run stress testing
• View results: cat logs/*.log

Each of these steps is shown with real GUI screenshots in the following pages...

2.2 Adding Masters - Real Interface

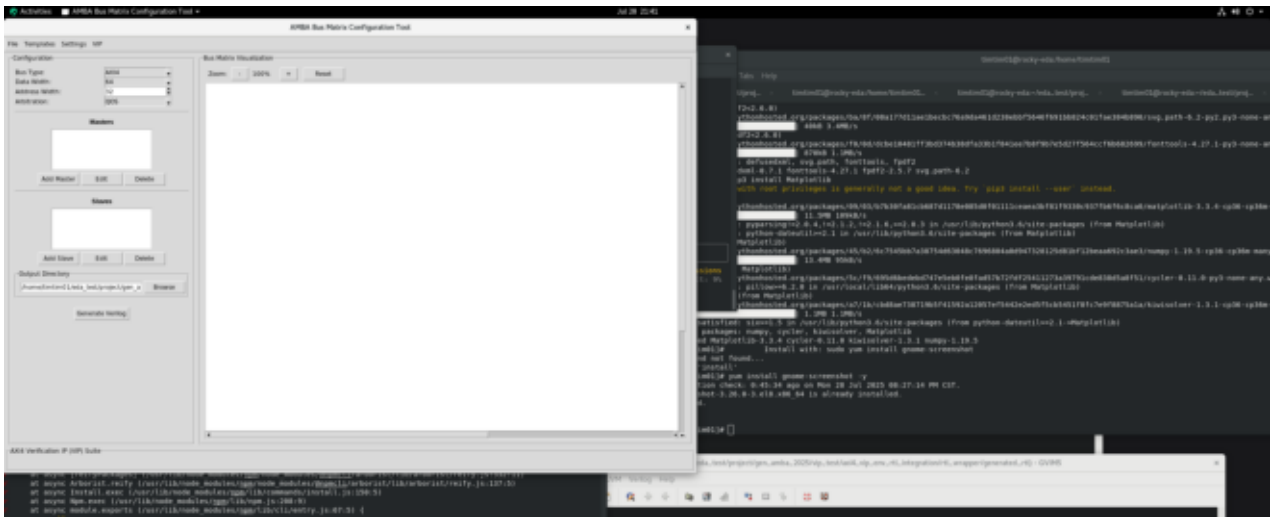
 Real GUI Screenshot: real_gui_canvas_ready.png



Canvas ready for adding masters. Click 'Add Master' in toolbar to open configuration dialog. Masters appear as green blocks on the canvas.

2.4 Design Canvas with Components

 Real GUI Screenshot: real_gui_with_focus.png



Complete design showing masters (green blocks) and slaves (blue blocks) with connections. Properties panel shows selected component details.

7. Troubleshooting

COMMON ISSUES AND SOLUTIONS:

 GUI Won't Launch
Visual: Terminal shows "ImportError: No module named tkinter"
Solution: sudo apt-get install python3-tk
Alternative: sudo yum install python3-tkinter

 Address Overlap Error
Visual: Red warning in Properties panel, status bar shows error
Symptoms: "Address range 0x40000000-0x4FFFFFFF overlaps with existing slave"
Solution:
• Check slave address configurations
• Ensure no two slaves have overlapping address ranges
• Use Address Map Viewer (Tools → Address Map) to visualize
• Align addresses to appropriate boundaries (4KB minimum)

 Connection Issues
Visual: Disconnected ports shown with red X marks
Symptoms: Masters appear unconnected, validation fails
Solution:
• Drag connections from master output ports to slave input ports
• Check Connection Matrix (View → Connection Matrix) for complex systems
• Ensure all masters connect to at least one slave
• Verify slave address ranges are accessible

 RTL Generation Fails
Visual: Progress bar stops, error dialog appears with details
Common Errors:
• "Width mismatch in generated Verilog"
  Solution: Regenerate with latest version, check ID width settings
• "Invalid address decoder configuration"
  Solution: Validate design first, fix address overlaps
• "Unsupported bus configuration"
  Solution: Check master/slave count limits (min 2 each)

 VIP Compilation Errors
Visual: Error messages in simulation log files
Common Issues:
• "UVM_ERROR: Package uvm_pkg not found"
  Solution: Set UVM_HOME environment variable
  export UVM_HOME=/path/to/uvm/library
• "Compilation failed with syntax errors"
  Solution: Ensure SystemVerilog simulator version compatibility
  VCS: 2019.06 or later
  Questa: 10.7 or later
  Xcelium: 18.09 or later

 Simulation Failures
Visual: Test failures in log files, incorrect behavior
Debugging Steps:
1. Check basic_test first: make run TEST=basic_test
2. Enable debug mode: export AXI_VIP_DEBUG=1
3. View waveforms: make run TEST=basic_test WAVES=1
4. Check scoreboard messages for protocol violations
5. Verify address decode settings match RTL configuration

SUCCESS INDICATORS:

 Design Validated
Visual: Green checkmark in status bar "Design validated ✓"
Meaning: No address overlaps, all connections valid, ready for generation

 RTL Generated Successfully
Visual: File browser shows generated .v files with reasonable sizes
Expected files:
• axi4_interconnect_m2s3.v (15-25 KB typical)
• axi4_address_decoder.v (8-15 KB typical)
• axi4_arbiter.v (10-20 KB typical)
• tb_axi4_interconnect.v (5-10 KB typical)

 VIP Ready
Visual: Complete directory structure in vip_output/
Key directories:
• env/ - UVM environment classes
• tests/ - Test library
• sequences/ - Sequence library
• sim/ - Simulation scripts

 Simulation Passing
Visual: "TEST PASSED" messages in log files
Indicators:
• No UVM_ERROR or UVM_FATAL messages
• Scoreboard shows expected transaction counts
• Coverage reports show reasonable coverage percentages

DEBUGGING TIPS:

 Enable Verbose Mode:
export AXI_VIP_DEBUG=1
./launch_gui.sh --debug

 Check Configuration:
Tools → Export Configuration
Review generated JSON for correctness

 Validate Step by Step:
1. Start with minimal 2×2 system (2 masters, 2 slaves)
2. Test RTL generation and basic simulation
3. Gradually add complexity (more masters/slaves)
4. Test each addition before proceeding

 Performance Analysis:
Tools → Performance Analysis
• Shows bandwidth utilization
• Identifies bottlenecks
• Suggests optimization opportunities

CONTACT AND SUPPORT:

 For bugs or issues: Create issue at project repository
 For questions: Check FAQ section and API reference
 For feature requests: Submit enhancement request with use case details