AMBA Bus Matrix Configuration Tool

User Guide and Reference Manual

Version 1.0.0
July 2025

Table of Contents

1. Introduction

The AMBA Bus Matrix Configuration Tool is a comprehensive solution for designing and implementing ARM AMBA-based System-on-Chip (SoC) interconnects. This tool provides both a graphical user interface for visual design and a powerful backend for generating synthesizable RTL and verification environments.

1.1 Key Features

• Visual bus matrix design with drag-and-drop interface

• Support for AXI4, AXI3, AHB, and APB protocols

• Automatic RTL generation with parameterizable configurations

• Complete UVM-based verification environment generation

• Built-in address overlap detection and validation

• Security and QoS configuration support

• Integration with industry-standard EDA tools

1.2 System Requirements

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

2. Getting Started

2.1 Installation

Clone the repository:

```
cd /your/project/directory
git clone <repository_url>
cd axi4_vip/gui
```

Install dependencies:

```
pip install -r requirements.txt
```

2.2 Launching the GUI

Start the GUI application:

```
./launch_gui.sh
# OR
python3 src/bus_matrix_gui.py
```

If GUI fails to launch:

• Check Python 3.6+ is installed: python3 --version

• Install tkinter if missing: sudo apt-get install python3-tk

• Make script executable: chmod +x launch_gui.sh

2.3 Setting Up Bus Matrix - Step by Step

Complete workflow to build a 2x3 system (CPU+DMA → DDR+SRAM+Peripherals):

STEP 1: Create New Project

• File → New Project (or Ctrl+N)

• Enter project name: "cpu_dma_system"

• Select bus type: AXI4 (recommended)

• Set data width: 64 bits (typical choice)

STEP 2: Add Masters

• Click "Add Master" button in left panel

• Master 1: Name="CPU_0", ID Width=4, Priority=2, QoS=Yes

• Click "Add Master" again

• Master 2: Name="DMA_0", ID Width=6, Priority=1, QoS=Yes

STEP 3: Add Slaves (CRITICAL - Address Configuration)

• Click "Add Slave" button

• Slave 1: Name="DDR_Memory", Base=0x00000000, Size=1GB

• Slave 2: Name="SRAM_Cache", Base=0x40000000, Size=256MB

• Slave 3: Name="Peripherals", Base=0x50000000, Size=256MB

STEP 4: Make Connections

• Drag from CPU_0 output port to each slave input port

• Drag from DMA_0 output port to DDR and SRAM (skip peripherals)

• OR use Connection Matrix: View → Connection Matrix

STEP 5: Validate Design

• Tools → Validate Design (or Ctrl+V)

• Fix any errors (typically address overlaps) before proceeding

STEP 6: Generate RTL

• Generate → Generate RTL (or Ctrl+G)

• Choose output directory (default: output_rtl/)

• Click Generate - creates .v files for synthesis

STEP 7: Generate VIP (Verification IP)

• Generate → Generate VIP (or Ctrl+Shift+G)

• Choose output directory (default: vip_output/)

• Creates complete UVM verification environment

STEP 8: Run VIP Simulation

```
cd vip_output/sim
make compile && make sim TEST=basic_test
view results: cat logs/basic_test.log
```

3. GUI Overview

3.1 Main Window Layout

The main window consists of several key areas: • Menu Bar: File operations, tools, and help • Toolbar: Quick access to common functions • Canvas: Main design area with grid • Properties Panel: Configure selected components • Status Bar: Current status and validation messages

3.2 Canvas Operations

Operation	Action
Pan	Middle-click and drag
Zoom	Scroll wheel or Ctrl +/-
Select	Left-click on component
Multi-select	Ctrl+click or drag box
Connect	Drag from master to slave
Delete	Select and press Delete
Properties	Double-click component

4. Creating Bus Designs

4.1 Adding Masters

Masters represent components that initiate transactions. Common examples include: • CPU cores • DMA engines • GPU processors • PCIe endpoints • Video codecs To add a master: 1. Click 'Add Master' button 2. Configure in the properties panel: - Name: Descriptive identifier - ID Width: Transaction ID bits (affects outstanding transactions) - Priority: Arbitration priority (higher wins) - QoS Support: Enable quality of service - Exclusive Support: Enable exclusive access

4.2 Adding Slaves

Slaves respond to transactions. Examples include: • Memory controllers (DDR, SRAM) • Peripheral registers • Configuration spaces • Bridge interfaces To add a slave: 1. Click 'Add Slave' button 2. Configure in the properties panel: - Name: Descriptive identifier - Base Address: Starting address (must be aligned) - Size: Address range in KB - Memory Type: Memory or Peripheral - Latency: Read/write cycle counts - Security: Access restrictions

5. RTL Generation

5.1 Generated Files

The RTL generator creates the following Verilog modules:

Filename	Description
axi4_interconnect_mNsM.v	Top-level interconnect module
axi4_address_decoder.v	Address decoding and routing logic
axi4_arbiter.v	Multi-master arbitration
axi4_router.v	Transaction routing logic
tb_axi4_interconnect.v	Basic testbench

5.2 Module Parameters

```
module axi4_interconnect_m2s3 #(
    parameter DATA_WIDTH = 128,
    parameter ADDR_WIDTH = 40,
    parameter ID_WIDTH   = 4,
    parameter USER_WIDTH = 1
)(
    input wire aclk,
    input wire aresetn,
    // Master and slave interfaces...
);
```

6. VIP Generation

6.1 Verification Environment Structure

```
vip_output/
■■■ env/              # UVM environment classes
■   ■■■ axi_env.sv
■   ■■■ axi_agent.sv
■   ■■■ axi_scoreboard.sv
■■■ tests/            # Test library
■   ■■■ axi_base_test.sv
■   ■■■ axi_basic_test.sv
■   ■■■ axi_stress_test.sv
■■■ sequences/        # Sequence library
■   ■■■ axi_base_sequence.sv
■   ■■■ axi_burst_sequence.sv
■   ■■■ axi_random_sequence.sv
■■■ tb/               # Testbench top files
■   ■■■ hvl_top.sv
■   ■■■ hdl_top.sv
■■■ sim/              # Simulation scripts
    ■■■ Makefile
    ■■■ run_test.sh
```

6.2 Running Simulations

Basic simulation flow:

```
cd vip_output/sim
make compile             # Compile design and testbench
make run TEST=axi_basic_test  # Run specific test
make run_all             # Run regression
```

7. Configuration Reference

7.1 Master Configuration

Parameter	Type	Default	Description
name	string	-	Master identifier
id_width	int	4	Transaction ID width
user_width	int	0	User signal width
priority	int	0	Arbitration priority
qos_support	bool	true	QoS enable
exclusive_support	bool	true	Exclusive access
default_prot	int	0b010	Default AxPROT
default_cache	int	0b0011	Default AxCACHE

7.2 Slave Configuration

Parameter	Type	Default	Description
name	string	-	Slave identifier
base_address	hex	-	Base address
size	int	-	Size in KB
memory_type	enum	Memory	Memory or Peripheral
read_latency	int	1	Read cycles
write_latency	int	1	Write cycles
num_regions	int	1	Protection regions
secure_only	bool	false	Secure access only

8. Advanced Features

8.1 Security Configuration

The tool supports ARM TrustZone security extensions: • Secure/Non-secure master designation • Per-slave security requirements • AxPROT[1] signal handling • Security violation detection and reporting Configure security in the Access Control Matrix.

8.2 QoS and Priority

Quality of Service features: • 4-bit QoS values per transaction • Priority-based arbitration • Weighted round-robin support • Latency-sensitive routing • Bandwidth allocation

8.3 Performance Optimization

The generated RTL includes several optimizations: • Registered outputs for timing closure • Parameterizable pipeline stages • Efficient arbitration logic • Minimal combinatorial paths • Clock domain crossing support

9. Troubleshooting

9.1 Common Issues

Issue	Solution
GUI won't launch	Check Python version and tkinter installation
Import errors	Verify all dependencies: pip install -r requirements.txt
Address overlap	Use address map viewer, check alignment
RTL syntax errors	Update to latest version, check generated files
VIP compile errors	Set UVM_HOME, check simulator version
Width mismatches	Regenerate with latest fixes, check ID_WIDTH

9.2 Debug Mode

Enable debug output:

```
export AXI_VIP_DEBUG=1
./launch_gui.sh --debug
```

10. API Reference

10.1 Command Line Interface

```
python3 src/bus_matrix_gui.py [options]

Options:
  --template FILE     Load template configuration
  --config FILE       Load saved configuration
  --output DIR        Set output directory
  --batch             Run in batch mode (no GUI)
  --generate-rtl      Generate RTL only
  --generate-vip      Generate VIP only
  --validate          Validate configuration only
  --debug             Enable debug output
  --version           Show version information
  --help              Show this help message
```

10.2 Python API

Example script:

```
from bus_config import BusConfig, Master, Slave
from axi_verilog_generator import AXIVerilogGenerator

# Create configuration
config = BusConfig()
config.data_width = 128
config.addr_width = 40

# Add master
master = Master("CPU")
master.id_width = 4
config.masters.append(master)

# Add slave
slave = Slave("Memory", 0x0, 1048576)
config.slaves.append(slave)

# Generate RTL
gen = AXIVerilogGenerator(config)
gen.generate()
```

Appendix A: AXI Protocol Overview

The AMBA AXI protocol is a high-performance, high-frequency protocol that supports: • Separate read and write channels • Multiple outstanding transactions • Out-of-order transaction completion • Burst transactions with only start address issued • Byte-lane strobes for partial writes • QoS signaling • Security extensions

AXI Channels:

Channel	Direction	Signals
Write Address	Master→Slave	AWID, AWADDR, AWLEN, AWSIZE, AWBURST, AWLOCK, AWCACHE, AWPROT, AWQC
Write Data	Master→Slave	WDATA, WSTRB, WLAST, WVALID, WREADY
Write Response	Slave→Master	BID, BRESP, BVALID, BREADY
Read Address	Master→Slave	ARID, ARADDR, ARLEN, ARSIZE, ARBURST, ARLOCK, ARCACHE, ARPROT, ARQOS, A
Read Data	Slave→Master	RID, RDATA, RRESP, RLAST, RVALID, RREADY

Appendix B: Example Configurations

B.1 Simple System (2 Masters, 3 Slaves)

```
{
  "bus_type": "AXI4",
  "data_width": 64,
  "addr_width": 32,
  "masters": [
    {"name": "CPU", "id_width": 4, "priority": 1},
    {"name": "DMA", "id_width": 4, "priority": 0}
  ],
  "slaves": [
    {"name": "RAM", "base_address": "0x00000000", "size": 1048576},
    {"name": "ROM", "base_address": "0x10000000", "size": 65536},
    {"name": "UART", "base_address": "0x20000000", "size": 4}
  ]
}
```

B.2 High-Performance Computing System

This example shows a complex SoC with multiple CPU clusters, GPU, and various memories: • 8 masters with different priorities and QoS requirements • 8 slaves including DDR controllers, caches, and peripherals • Security zones and access control • Optimized for high bandwidth and low latency