

# Verilog Circuit Visualizer - User Guide

## Table of Contents

- [\[Introduction\]\(#introduction\)](#)
- [\[Prerequisites\]\(#prerequisites\)](#)
- [\[Installation\]\(#installation\)](#)
- [\[Required Files\]\(#required-files\)](#)
- [\[Step-by-Step Usage\]\(#step-by-step-usage\)](#)
- [\[Input File Format\]\(#input-file-format\)](#)
- [\[Understanding Output Files\]\(#understanding-output-files\)](#)
- [\[Examples\]\(#examples\)](#)
- [\[Troubleshooting\]\(#troubleshooting\)](#)
- [\[Quick Reference\]\(#quick-reference\)](#)

## Introduction

The Verilog Circuit Visualizer is a Python tool that helps you visualize and simulate digital circuits described in Verilog. It generates PDF diagrams showing your circuit structure and how signals flow through it with logic values (0, 1, or X for unknown).

### What this tool does:

- Parses your Verilog circuit files
- Generates visual circuit diagrams (PDF format)
- Simulates logic values through your circuit
- Shows signal flow from inputs to outputs

### Who can use this:

- Students learning digital design
- Engineers debugging Verilog circuits

- Anyone who needs to visualize circuit behavior

## Prerequisites

Before using this tool, you need:

- **Python 3.7 or higher** installed on your computer
- Check by running: `python --version` or `python3 --version`
- Download from: <https://www.python.org/downloads/>
- **Graphviz** installed on your system
- **Windows:** Download installer from <https://graphviz.org/download/>
- After installation, restart your terminal/command prompt
- Verify: Run `dot -V` in terminal
- **Linux:** `sudo apt-get install graphviz`
- **macOS:** `brew install graphviz`
- **Required Python packages** (installed via requirements.txt)

## Installation

### Step 1: Get the Code

If you haven't already, download or clone the repository:

```
git clone https://github.com/neil066/New-VLSI-Code.git cd New-VLSI-Code
```

### Step 2: Install Python Dependencies

Open a terminal/command prompt in the project directory and run:

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

This installs:

- `pyverilog` - For parsing Verilog files
- `graphviz` - Python interface for Graphviz
- `pydot` - For DOT file manipulation
- `reportlab` - For PDF generation (optional)

## Step 3: Verify Installation

Test that everything works:

```
python verilog_visualizer.py --help
```

You should see help text. If you get errors, check the Troubleshooting section.

## Required Files

To use this program, you need **two essential files**:

### 1. Verilog File (.v)

**What it is:** Your circuit description written in Verilog

**Requirements:**

- Must be a valid Verilog file with `.v` extension
- Should contain at least one module definition
- Can be hierarchical (with submodules) or flat (mapped)
- Must use supported gate types (see below)

**Supported gate types:**

- Basic gates: `and`, `or`, `not`, `nand`, `nor`, `xor`, `xnor`
- Arithmetic: `FA` (Full Adder), `HA` (Half Adder), `FS` (Full Subtractor), `HS` (Half Subtractor)
- Multiplexers: `MUX2`, `MUX4`
- Custom hierarchical modules

### Example Verilog file structure:

```
module my_circuit(input a, input b, input c, output y); wire w1, w2; and
G1(w1, a, b); or G2(w2, w1, c); not G3(y, w2); endmodule
```

### Where to put it:

- Can be anywhere on your computer
- Common locations: Verilog Files/ directory or your project folder
- Use full path or relative path when running the program

## 2. Input File (.txt) - Optional but Recommended

**What it is:** A text file containing the input values for your circuit

### Why you need it:

- The program needs to know what values to apply to your circuit inputs
- Without it, you'll be prompted to enter values manually (tedious for many inputs)
- Makes testing easier and reproducible

### Requirements:

- Must contain values for **ALL** primary inputs in your Verilog file
- Must match the number and names of inputs exactly
- Can be in different formats (see Input File Format section)

### Example input file:

```
a=1, b=0, c=1
```

### Where to put it:

- Can be anywhere on your computer
- Common location: Input Files/ directory
- Use full path or relative path when prompted

## Step-by-Step Usage

### Basic Workflow

- **Prepare your files**

- Have your Verilog file ready (e.g., `my_circuit.v`)
- Create an input file with test values (e.g., `my_inputs.txt`)

- **Run the program**

```
python verilog_visualizer.py my_circuit.v -f
```

or

```
python verilog_visualizer.py my_circuit.v -s
```

- **Choose simulation mode**

- `-f` or `--full`: Shows all values at once (faster, recommended)
- `-s` or `--step`: Shows values level by level (good for debugging)

- **Provide input file when prompted**

```
Enter input file path (or type 'n' for no input file): Input  
Files/my_inputs.txt
```

- Type the path to your input file
- Or type `n` to enter values manually

- **Wait for processing**

- Program parses your Verilog file
- Generates structure diagram
- Runs simulation
- Generates simulation diagram

- **Find your output files**

- Look for `{filename}_structure.pdf` and `{filename}_simulation.pdf`
- Open them to see your circuit visualization

## Detailed Step-by-Step Example

Let's say you have a file called `adder.v`:

**Step 1:** Open terminal in the project directory

**Step 2:** Run the command:

```
python verilog_visualizer.py "Verilog Files/adder.v" -f
```

**Step 3:** You'll see:

```
=== Verilog Circuit Visualizer === Parsing Verilog file(s): Verilog
Files/adder.v
```

**Step 4:** When prompted for input file:

```
Enter input file path (or type 'n' for no input file): Input
Files/test_adder.txt
```

**Step 5:** Program processes and shows:

```
Loading input values from: Input Files/test_adder.txt Generating structure
visualization... Generating simulation visualization...
```

**Step 6:** Check for output files:

- adder\_structure.pdf - Circuit structure without values
- adder\_simulation.pdf - Circuit with logic values

## Input File Format

The input file tells the program what values to apply to your circuit inputs. You can use **two formats**:

### Format 1: Comma-Separated Assignments (Recommended)

**Syntax:** input\_name=value, input\_name=value, ...

**Example:**

```
a=1, b=0, c=1, d=0
```

**For multi-bit inputs:**

```
a[0]=1, a[1]=0, a[2]=1, a[3]=0, b[0]=1, b[1]=1, cin=0
```

**Rules:**

- Each assignment: name=value
- Separate assignments with commas
- Values can be: 0 (LOW), 1 (HIGH), or x (unknown)
- Spaces around = and commas are optional

- Must include ALL primary inputs from your Verilog file

**Complete example file (`test\_adder.txt`):**

```
a[0]=1, a[1]=0, a[2]=1, a[3]=0, b[0]=1, b[1]=1, b[2]=0, b[3]=1, cin=0
```

## Format 2: Space-Separated Values

**Syntax:** Values separated by spaces, in the order inputs appear in Verilog

**Example:**

```
1 0 1 0
```

**Rules:**

- Values must be in the same order as inputs in your Verilog file
- Separate with spaces
- Values: 0, 1, or x
- Less flexible than Format 1 (order-dependent)

## How to Determine Required Inputs

**Method 1: Check your Verilog file**

- Look for `input` declarations in your module
- Example: `module adder(input a, input b, input cin, output sum);`
- Inputs: `a`, `b`, `cin` (3 inputs needed)

**Method 2: Run the program without input file**

- Type `n` when prompted for input file
- Program will list all detected inputs:

```
Circuit analysis detected 3 primary input(s): a, b, cin
```

**Method 3: Check example files**

- Look in `Input Files/` directory for examples
- Match the pattern to your circuit

## Common Mistakes

**[ERROR] Wrong number of inputs:**

```
# Verilog has: a, b, c (3 inputs) # Input file has: a=1, b=0 (only 2 inputs) - WRONG!
```

**[OK] Correct:**

```
a=1, b=0, c=1 # All 3 inputs provided
```

**[ERROR] Wrong input names:**

```
# Verilog has: input_a, input_b # Input file has: a=1, b=0 - WRONG! (names don't match)
```

**[OK] Correct:**

```
input_a=1, input_b=0 # Names match Verilog exactly
```

**[ERROR] Invalid values:**

```
a=2, b=high, c=low # WRONG! Only 0, 1, or x allowed
```

**[OK] Correct:**

```
a=1, b=0, c=x # Valid values
```

## Understanding Output Files

After running the program, you'll get **four files** for each Verilog file:

### 1. `{filename}\_structure.dot`

- **What it is:** Graphviz DOT format file (text format)
- **Purpose:** Intermediate file showing circuit structure
- **When to use:** For debugging or manual editing
- **Can open with:** Text editor or Graphviz tools

### 2. `{filename}\_structure.pdf`

- **What it is:** PDF diagram of your circuit structure
- **Shows:** Circuit topology, gates, connections (NO logic values)

- **Use when:** You want to see the circuit layout
- **Colors:**
  - Green ellipses = Primary inputs
  - Red ellipses = Primary outputs
  - Boxes = Logic gates

### 3. `{filename}\_simulation.dot`

- **What it is:** Graphviz DOT format with simulation values
- **Purpose:** Intermediate file with logic values
- **When to use:** For debugging simulation issues

### 4. `{filename}\_simulation.pdf` ■ **\*\*Most Important\*\***

- **What it is:** PDF diagram with logic values propagated
- **Shows:** Circuit with all wire values (0, 1, or X)
- **Use when:** You want to verify circuit behavior
- **Features:**
  - Shows input values (green)
  - Shows output values (red)
  - Shows intermediate wire values
  - Values displayed on connections

## Example Output File Names

If you process `adder.v`:

- `adder_structure.dot`
- `adder_structure.pdf`
- `adder_simulation.dot`
- `adder_simulation.pdf`

# Examples

## Example 1: Simple Gates Circuit

**Verilog file (`simple\_gates.v`):**

```
module simple_gates(input a, input b, input c, output y); wire w1, w2; and
G1(w1, a, b); or G2(w2, w1, c); not G3(y, w2); endmodule
```

**Input file (`test\_simple.txt`):**

```
a=1, b=0, c=1
```

**Command:**

```
python verilog_visualizer.py "Verilog Files/simple_gates.v" -f
```

**When prompted:**

```
Enter input file path: Input Files/test_simple.txt
```

**Output:**

- simple\_gates\_structure.pdf - Shows 3 gates connected
- simple\_gates\_simulation.pdf - Shows: a=1, b=0, c=1 → w1=0, w2=1, y=0

## Example 2: 4-Bit Adder (Hierarchical)

**Verilog file:** 4-bit-add-hier.v (uses Full Adder and Half Adder modules)

**Input file (`test\_4bit\_adder.txt`):**

```
a[0]=1, a[1]=0, a[2]=1, a[3]=0, b[0]=1, b[1]=1, b[2]=0, b[3]=1, cin=0
```

**Command:**

```
python verilog_visualizer.py "Verilog Files/4-bit-add-hier.v" -s
```

**Note:** Using `-s` (step mode) for hierarchical circuits helps see level-by-level propagation.

## Example 3: Multiple Files

**Command:**

```
python verilog_visualizer.py "Verilog Files/circuit1.v" "Verilog Files/circuit2.v" -f
```

**Result:** Generates output files for both circuits.

## Troubleshooting

### Problem: "Error: Verilog file 'X' not found"

**Cause:** File path is incorrect or file doesn't exist

**Solutions:**

- Check file path spelling
- Use quotes if path has spaces: "Verilog Files/my circuit.v"
- Use full path: C:/Users/YourName/circuit.v
- Check file extension is .v

### Problem: "You must specify a simulation mode"

**Cause:** Missing -f or -s flag

**Solution:** Always include one:

```
python verilog_visualizer.py circuit.v -f # or -s
```

### Problem: "graphviz not found" or "Executable not found"

**Cause:** Graphviz not installed or not in PATH

**Solutions:**

- **Windows:**
  - Reinstall Graphviz from <https://graphviz.org/download/>
  - Restart terminal after installation
  - Add to PATH manually if needed

- **Linux/macOS:**
- Run: `sudo apt-get install graphviz` (Linux)
- Run: `brew install graphviz` (macOS)
- Verify: Run `dot -V` in terminal

## Problem: "pyverilog not installed"

**Cause:** Python dependencies not installed

**Solution:**

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

## Problem: Input file not found

**Cause:** Wrong path to input file

**Solutions:**

- Use relative path: `Input Files/test.txt`
- Use full path: `C:/Users/YourName/Input Files/test.txt`
- Check file exists before running
- Use forward slashes `/` or double backslashes `\\` in paths

## Problem: "No modules found in Verilog files"

**Cause:** Verilog file has syntax errors or no valid modules

**Solutions:**

- Check Verilog syntax
- Ensure file has `module ... endmodule` structure
- Verify file is valid Verilog (test in simulator)
- Check for typos in module declarations

## Problem: Empty or incorrect PDF

**Cause:** Circuit has issues or inputs not provided correctly

**Solutions:**

- Check input file has correct number of inputs
- Verify input names match Verilog exactly
- Check `.dot` files to see circuit structure
- Ensure all inputs have valid values (0, 1, or x)

## Problem: Values not propagating correctly

**Cause:** Input file format issue or missing inputs

**Solutions:**

- Verify all inputs provided in input file
- Check input file format (comma-separated assignments)
- Ensure input names match Verilog exactly (case-sensitive)
- Try manual input entry to isolate the issue

## Problem: "Invalid input" during manual entry

**Cause:** Entered invalid value

**Solution:** Only enter: 0, 1, x, or press Enter (for unknown)

# Quick Reference

## Command Syntax

```
python verilog_visualizer.py <verilog_file(s)> [OPTIONS]
```

## Required Options

- `-f` or `--full`: Full simulation (all values at once)

- `-s` or `--step`: Step-by-step simulation (level by level)

**You must use one of these!**

## Common Commands

```
# Single file, full simulation python verilog_visualizer.py circuit.v -f #  
Single file, step simulation python verilog_visualizer.py circuit.v -s #  
Multiple files python verilog_visualizer.py file1.v file2.v -f # With path  
python verilog_visualizer.py "Verilog Files/circuit.v" -f
```

## Input File Format Quick Reference

### Format 1 (Recommended):

```
input1=value1, input2=value2, input3=value3
```

### Format 2:

```
value1 value2 value3
```

**Valid values:** 0 (LOW), 1 (HIGH), x (unknown)

## Output Files

For `circuit.v`, you get:

- `circuit_structure.pdf` - Structure diagram
- `circuit_simulation.pdf` - Simulation with values ■

## Getting Help

```
python verilog_visualizer.py --help
```

## Example Files Location

- Verilog files: `Verilog Files/` directory
- Input files: `Input Files/` directory

- Test guide: `TESTING_GUIDE.md`

## Summary Checklist

Before running the program, make sure you have:

- ☐ Python 3.7+ installed
- ☐ Graphviz installed and in PATH
- ☐ Dependencies installed (`pip install -r requirements.txt`)
- ☐ Verilog file (.v) ready
- ☐ Input file (.txt) with correct number of inputs (optional but recommended)
- ☐ Know which simulation mode to use (`-f` or `-s`)

When running:

- ☐ Use correct command syntax with `-f` or `-s`
- ☐ Provide correct path to Verilog file
- ☐ Provide correct path to input file when prompted (or type `n`)
- ☐ Wait for processing to complete
- ☐ Check output PDF files

## Additional Resources

- **README.md**: General project information
- **TESTING\_GUIDE.md**: Detailed testing examples
- **GitHub Repository**: <https://github.com/neil066/New-VLSI-Code>
- **Graphviz Documentation**: <https://graphviz.org/documentation/>

**Happy Visualizing! ■**

If you encounter issues not covered here, check the troubleshooting section or examine the generated `.dot` files for debugging.