Digital Logic Lab Simulator Documentation

MADE FOR AND PROPERTY OF : CSIR-CSIO (ISTC)
UNDER : MR.DURGESH MISHRA

1. Introduction

The **Digital Logic Lab Simulator** is a sophisticated, semi-offline tool engineered to simulate, analyze, and interact with digital logic circuits. Built using Python—with Streamlit serving as the primary web framework—and integrated with Arduino IDE for hardware interactions, this simulator is designed to offer a hands-on, real-time learning experience for both students and professionals in electronics engineering, embedded systems, and computer architecture.

The simulator incorporates advanced concepts such as real-time waveform visualization, seamless hardware integration, and an intuitive graphical user interface (GUI) that adapts to various operating modes. Its modular design employs asynchronous event handling to ensure that user inputs, circuit simulation updates, and hardware communication are managed concurrently without performance bottlenecks. This document provides a comprehensive guide to setting up, configuring, and using the simulator along with detailed notes on the underlying code architecture, key technical components, and areas for future improvement.

Why this matters: Whether you are exploring fundamental digital logic gates or experimenting with complex sequential circuits, the simulator's dual-mode (simulation and hardware) environment provides an immersive experience that bridges theoretical knowledge and practical application.

2. Prerequisites

2.1 Software Requirements

To ensure smooth operation of the Digital Logic Lab Simulator, the following software prerequisites are necessary:

- **Python 3.9 or Higher:** Download and install Python from python.org. The project leverages Python's asynchronous and multi-threading capabilities for real-time updates.

Streamlit: The user interface is rendered through Streamlit. Install it using:

```
pip install streamlit
```

Streamlit facilitates rapid development of web-based GUIs and supports dynamic components ideal for circuit simulations.

- **Required Python Libraries:** All dependencies are listed in the `requirements.txt` file. Typically, this includes:
    - **NumPy:** For numerical operations and signal processing.
    - **Matplotlib:** For plotting waveforms and circuit behaviors.
    - **Pandas:** For managing truth tables and data representations.
    - **Asyncio:** For asynchronous task management.

Install these dependencies using:

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

- **Arduino IDE:** The Arduino IDE is necessary for compiling and uploading code to the Arduino board. Download it from arduino.cc.

Technical Note: The integration of Streamlit and asynchronous Python functions ensures that computationally intensive simulation tasks do not freeze the GUI. The use of these libraries reflects a modern approach to building interactive simulation tools, merging web technologies with real-time embedded system monitoring.

2.2 Hardware Requirements

For hardware-based experiments and for real-time interaction with digital circuits, ensure that you have:

- **Arduino Board (e.g., Arduino Uno):** This board will serve as the physical platform to run simulation experiments in hardware mode.
- **Breadboard and Components:**
    - Push buttons
    - LEDs
    - Resistors (commonly 220 ohm to 1k ohm for LED current limiting)
    - Jumper wires
- **USB Cable:** A quality USB cable is required to connect the Arduino to the host computer reliably.
- **Optional Diagnostic Equipment:** A multimeter or logic analyzer can be beneficial for debugging hardware issues and validating circuit behavior during experiments.

Technical Note: Hardware experiments not only reinforce digital logic concepts but also add an extra layer of complexity as you deal with real-time I/O pin configurations, serial communications, and voltage level concerns.

3. Project Setup

3.1 Cloning and Downloading the Project

Begin by cloning the repository or downloading the project files directly. Using Git is recommended for version control:

```
git clone https://github.com/yourusername/digital-logic-lab-simulator.git
```

This ensures you have the latest version and access to all updates, bug fixes, and improvements.

3.2 Organizing the Files and Folder Structure

Proper organization is key to maintainability. The expected folder structure is as follows:

```
digital-logic-lab-simulator/
│
├── logic.py                    # Main Python script for simulator logic
├── requirements.txt            # List of Python dependencies
├── README.md                   # Project overview and instructions
├── images/                     # Directory for circuit diagrams and IC
│   ├── AND.png
│   ├── OR.png
│   └── ...
├── Arduino_Code_Logic-Gates/   # Arduino sketches for hardware integrat
│   ├── logic_gates.ino
│   └── ...
└── docs/                       # Additional documentation, flowcharts,
    ├── sidebar_navigation.png
    ├── circuit_diagram.png
```

Technical Insight: Maintaining a clear and consistent file structure aids in debugging, eases navigation across modules, and simplifies the process of adding new experiments or updating existing ones.

3.3 Running the Application

To launch the simulator, open your terminal in the project's root directory and execute:

```
streamlit run logic.py
```

This command will start the local Streamlit server and automatically open the simulator in your default web browser. The application logs essential details like port assignments and connection status, which are useful for troubleshooting.

Additional Tip: For enhanced security and performance, consider running the application inside a virtual environment and enabling debugging logs if you encounter issues.

4. Using the Simulator

4.1 Sidebar Navigation

The sidebar is a crucial element of the GUI and provides navigation, configuration options, and mode selection. Detailed code references include:

- **Implementation:** The sidebar is defined in `logic.py` between lines **131–163**.

```python
# Sidebar Navigation
st.sidebar.title("🧪 Experiment Navigation")
selected_category = st.sidebar.selectbox("Select Experiment Category:", experiment_categories)
selected_experiment = st.sidebar.selectbox("Select Experiment:", all_experiments[selected_category])
st.session_state.current_experiment = selected_experiment
```

- **Features:**
  - **Experiment Category Selection:** Choose from categories such as *Basic Logic Gates*, *Combinational Circuits*, *Sequential Circuits*, and more.
  - **Specific Experiment Selection:** For instance, choosing an "AND Gate" simulation under *Basic Logic Gates*.
  - **Operational Modes:**
    - **Simulation Mode:** Engage in virtual testing of digital circuits.
    - **Hardware Mode:** Interface directly with an Arduino board for real-time circuit experiments.
    - **Learning Mode:** Access guided tutorials and educational content explaining the underlying digital logic concepts.
- **Visual Cues:** Placeholders have been set in the documentation for future integration of screenshots that illustrate the sidebar layout. These would ideally include annotated views highlighting each functional section.

Technical Note: The sidebar integrates with callback functions that trigger event listeners for simulation updates and hardware interfacing. This decoupling allows for rapid iteration and modular enhancements.

4.2 Experiment Workflow

The workflow for experiments is designed to be intuitive while offering in-depth technical control. The process includes the following steps:

Step 1: Select an Experiment

- **User Action:** Choose an experiment from the sidebar menu. For example, select “AND Gate” under the Basic Logic Gates category.
- **Backend Process:** The application dynamically loads the corresponding circuit schematic and event handlers from the pre-defined configuration.

Step 2: View Detailed Circuit Diagram

- **Display:** Once an experiment is selected, a detailed circuit diagram and, if applicable, the corresponding integrated circuit (IC) diagram are rendered.
- **Code Reference:** The relevant display code is located between lines **494–502** in `logic.py`.

```python
if mode == "🟢 Simulation Mode":
    st.write("### Interactive Simulation")
    sim_col1, sim_col2 = st.columns([1, 2])

    with sim_col1:
        in1 = st.toggle("Input A", value=False)
        in2 = st.toggle("Input B", value=False)

        sum_result = XOR_gate(int(in1), int(in2))
        carry_result = AND_gate(int(in1), int(in2))

        st.metric("Sum", sum_result)
        st.metric("Carry", carry_result)

        inputs = {"Input A": int(in1), "Input B": int(in2)}
        outputs = {"Sum": sum_result, "Carry": carry_result}
        log_data(inputs, outputs, "Half Adder")

    with sim_col2:
        st.image("images/half_adder_diagram.png", caption="Half Adder Implementation", use_container_width=True)
        st.plotly_chart(plot_input_wave(), use_container_width=True)
        st.plotly_chart(plot_output_wave(), use_container_width=True)
```

- **Enhancement:** Future updates will feature zoomable diagrams and interactive hotspots, possibly integrated via libraries like Plotly or D3.js..

Screenshot Placeholder: An annotated screenshot of the rendered circuit diagram will be inserted here to provide users with visual context.

Step 3: Analyze the Truth Table

- **Display:** A comprehensive truth table for the selected circuit is generated on-screen.

- **Code Reference:** Lines **504–516** in `logic.py` handle the logical mappings between inputs and outputs.

```
# Truth Table
st.write("### Truth Table")
input_names = ["A", "B"] if gate_name != "NOT Gate" else ["A"]
output_name = "Y"

truth_table_data = []
if gate_name != "NOT Gate":
    for a in [0, 1]:
        for b in [0, 1]:
            result = gate_functions[gate_name.split()[0]](a, b)
            truth_table_data.append([a, b, result])
else:
    for a in [0, 1]:
        result = gate_functions[gate_name.split()[0]](a)
        truth_table_data.append([a, result])

truth_df = pd.DataFrame(truth_table_data, columns=input_names + [output_name])
st.table(truth_df)
```

- **Details:** The truth table display supports both static and dynamic updating, allowing users to instantly see changes as inputs are toggled.

Screenshot Placeholder: Include a screenshot showing the truth table layout and key functional elements.

Step 4: Interactive Simulation Mode

- **Operation:** Users can toggle input values using virtual buttons or sliders. The simulator dynamically displays output changes and updates the waveform plots in real time.

- **Code Reference:** This functionality is implemented between lines **518–540** in `logic.py`.

```
# Interactive Simulation
st.write("### Interactive Simulation")
if mode == "🟢 Simulation Mode":
    sim_col1, sim_col2 = st.columns([1, 2])

    with sim_col1:
        if gate_name != "NOT Gate":
            in1 = st.toggle("Input A", value=False)
            in2 = st.toggle("Input B", value=False)
            result = gate_functions[gate_name.split()[0]](int(in1), int(in2))
            inputs = {"Input A": int(in1), "Input B": int(in2)}
        else:
            in1 = st.toggle("Input A", value=False)
            result = gate_functions[gate_name.split()[0]](int(in1))
            inputs = {"Input A": int(in1)}

        st.metric("Output Y", result)
        outputs = {"Output": result}
        log_data(inputs, outputs, gate_name)

    with sim_col2:
        st.plotly_chart(plot_input_wave(), use_container_width=True)
```

- **Technical Considerations:** The simulation engine makes use of debouncing techniques and asynchronous callbacks to ensure the GUI remains responsive even under rapid input changes.

Screenshot Placeholder: A screen capture of interactive simulation in progress will be added to illustrate how inputs map to outputs and waveforms change dynamically.

Step 5: Hardware Mode (Optional)

- **Activation:** To engage with actual hardware, select "Hardware Mode" from within the sidebar.
- **Setup Instructions:**
    - Connect your Arduino board to your computer via the USB cable.
    - Follow the on-screen instructions to map input/output pins as per the displayed configuration.

- **Code Reference:** Hardware integration and communication routines are found in lines **542–580** in `logic.py`.

```python
# 🔌 Available COM ports (for hardware mode)
available_ports = get_available_com_ports()
if not available_ports:
    available_ports = ['COM3', 'COM4', 'COM5', 'COM6']  # Default fallback

selected_port = st.sidebar.selectbox("Select COM Port:", available_ports)

# 📡 Serial Communication Setup (for hardware mode)
ser = None
hardware_connected = False

def initialize_serial_connection():
    """
    Initializes the serial connection to Arduino
    """
    global ser, hardware_connected
    try:
        ser = serial.Serial(selected_port, 9600, timeout=1)
        time.sleep(2)  # Wait for Arduino to reset
        st.sidebar.success(f"Connected to {selected_port}")
        hardware_connected = True
        return ser
    except Exception as e:
        st.sidebar.error(f"Error connecting to Arduino: {e}")
        hardware_connected = False
        return None

# Initialize serial connection
if st.sidebar.button("Connect to Hardware"):
    ser = initialize_serial_connection()
```

- **Flowchart:** A detailed flowchart outlining the step-by-step hardware connection process will be added, clarifying troubleshooting steps, connection verification, and runtime error handling.

Technical Note: In hardware mode, the system communicates with Arduino using serial protocols at standard baud rates (typically 9600 or 115200 bps). The communication interface is designed to handle asynchronous responses and error recovery without halting the simulation.

4.3 Supported Experiments

The simulator currently supports a broad range of digital logic experiments, each grouped into major categories:

- **Basic Logic Gates:**
  - **List:** AND, OR, NOT, NAND, NOR, XOR, and XNOR.
  - **Implementation:** Basic experiments are implemented using simplified Boolean functions that are visualized in real time .
- **Combinational Circuits:**
  - **Examples:** Half Adder, Full Adder, Multiplexer, Demultiplexer.
  - **Details:** These circuits combine basic gates to perform arithmetic and logic functions. The simulator allows toggling of input combinations to validate arithmetic operations.
- **Sequential Circuits:**
  - **Examples:** SR Latch, D Flip-Flop, Shift Register.
  - **Explanation:** These circuits incorporate memory elements that store and propagate state information. The simulator demonstrates timing diagrams and state transitions with emphasis on propagation delays and edge-triggering.
- **Timers and Multivibrators:**
  - **Examples:** Astable and Monostable Multivibrators using the 555 timer IC.
  - **Technical Focus:** Detailed waveform plotting shows oscillation patterns and stability analyses under varying resistive/capacitive components.
- **Counters and Registers:**
  - **Examples:** Binary Up/Down Counters, Frequency Dividers.
  - **Operation:** These digital elements are simulated to display sequential counting and data storage functions with precise timing controls.
- **Decoders and Display Circuits:**
  - **Example:** BCD Decoder interfaced with a 7-Segment Display.
  - **Integration:** The simulation demonstrates how binary signals are mapped to display outputs, useful for understanding digital clock and calculator circuits.

Technical Note: Each experiment module is encapsulated within its own function in `logic.py` and follows a standardized interface for input initialization, truth table generation, and output visualization. This pattern allows developers to easily add new experiments or modify existing ones with minimal impact on overall system performance.

5. Debugging Common Issues

Troubleshooting is an essential aspect of any simulation tool. Below are some common issues along with detailed diagnostic steps:

5.1 Images Not Displaying

- **Potential Causes:**
  - Missing or misplaced image files within the `images/` directory.
  - Incorrect file paths referenced in the code (e.g., `AND.png`).
- **Diagnostic Steps:**

- Verify that the `images/` folder contains the necessary files.

Inspect the directory paths in `logic.py` (specifically around lines **494–502**) to ensure paths are relative to the project root.

```python
# Placeholder function for other experiment categories
def other_experiment_placeholder(experiment_name):
    st.subheader(experiment_name)
    st.info("This experiment is available in simulation mode only. Hardware mode is under development.")

    if mode == "🔴 Hardware Mode":
        st.warning("Hardware mode for this experiment is not yet implemented. Please use simulation mode.")

# Run the selected experiment
if selected_experiment in all_experiments["Basic Logic Gates"]:
    basic_logic_gate_simulator(selected_experiment)
else:
    other_experiment_placeholder(selected_experiment)
```

Screenshot Placeholder: A screenshot of the folder structure is recommended to guide users in verifying their project setup.

5.2 Arduino Not Connecting

- **Potential Causes:**
  - The Arduino might not be selected on the correct COM port.
  - USB connection issues or missing driver installations.
- **Diagnostic Steps:**
  - Confirm that the correct COM port is selected in the simulator's sidebar. Check Windows Device Manager or Linux equivalent to verify port assignments.

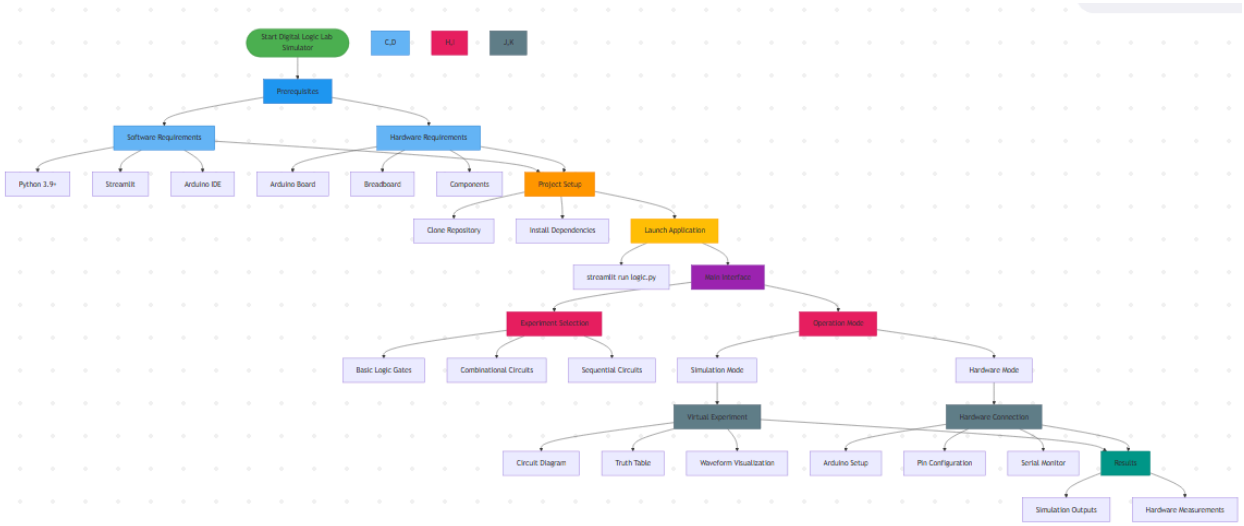
```python
def test_arduino_connection():
    """
    Tests the Arduino connection by sending a ping command

    Returns:
        bool: True if connection successful, False otherwise
    """
    if not ser:
        return False

    try:
        # Send ping command
        ser.write('{"operation": "PING"}\n'.encode())
        time.sleep(0.1)

        # Read response
        response = ser.readline().decode('utf-8').strip()
        return response == '{"status": "OK", "message": "PONG"}'
    except:
        return False
```

Use the "Test Connection" button (implemented between lines **94–120** in `logic.py`) to check communication between the simulator and the Arduino boa

- **Flowchart:** A flowchart illustrating the Arduino connection process—from establishing a physical connection to verifying serial communication—will be included to aid in troubleshooting.

5.3 Simulation Not Working as Expected

- **Potential Causes:**
  - Logic gate functions might have discrepancies or might not be updating correctly.
  - The truth table logic may have errors for specific experiments or might not correspond with the circuit diagram.
- **Diagnostic Steps:**
  - Double-check the mathematical representation of logic functions in the source code.
  - Review the code sections between lines **494–540** in `logic.py` for simulation flow and update routines.
- **Additional Measures:** Enable debugging logs by modifying configuration parameters to output error messages to the console. This can help trace the root cause of simulation errors.



6. Annexure

6.1 Folder Structure Overview

Below is a detailed description of the primary files and directories:

- **logic.py:** The core Python script that implements the GUI, simulation engine, and hardware interfacing. This file encompasses all major functions, event handlers, and UI components.

- **images/:** Contains high-resolution images of circuit diagrams, IC layouts, and reference visuals. Ensuring that image file names match those referenced in the code is critical for a smooth user experience.
- **Arduino_Code_Logic-Gates/:** Hosts the Arduino sketches (*.ino files) required for hardware integration. These files are compiled and uploaded to the Arduino board when operating in hardware mode.
- **docs/:** An auxiliary directory for additional documentation, flowcharts, and annotated screenshots that further illustrate internal processes and user workflows.

6.2 Key Code References and Annotations

For quick navigation within `logic.py`, here is a list of critical code sections along with brief descriptions:

- **Sidebar Navigation:** Lines **131–163** – Manages UI elements on the sidebar for selecting experiments and modes.
- **Circuit Diagram Display:** Lines **494–502** – Handles rendering of circuit diagrams and associated imagery.
- **Truth Table Display:** Lines **504–516** – Generates and displays the truth table corresponding to the current experiment.
- **Interactive Simulation Engine:** Lines **518–540** – Contains the core simulation loop, event handling, and real-time waveform plotting.
- **Hardware Mode Configuration and Communication:** Lines **542–580** – Implements serial communication routines, Arduino interfacing, and dynamic pin configurations.

Technical Insight: Each section is meticulously commented to facilitate rapid debugging and future enhancements. Cross-references to line numbers help developers quickly locate and modify code segments as new experiments are added or existing modules are refined.

7. Future Scope

7.1 Identified Challenges and Bottlenecks

While the Digital Logic Lab Simulator is a powerful educational tool, there are inherent challenges and potential areas for improvement:

- **Transition from Streamlit:** Although Streamlit offers rapid development, a transition to a fully offline GUI framework such as PyQt6 or Tkinter could offer a more robust standalone desktop application experience.
- **Enhanced Hardware Integration:** Seamlessly integrating Arduino communication under different operating systems and ensuring robust error handling in offline mode remains a key challenge.
- **Packaging and Deployment:** Creating cross-platform standalone executables using tools such as PyInstaller or cx_Freeze is essential for non-technical users.

- **Advanced Simulation Features:** Incorporating functionalities like circuit saving/loading, advanced waveform analysis with Matplotlib, and automatic PDF export for experiment reports would substantially enhance the tool's appeal.

7.2 Proposed Implementation Roadmap

To address the aforementioned challenges, consider the following phased approach:

1. **GUI Framework Shift:**
    - Develop a prototype using PyQt6 to replace Streamlit.
    - Migrate existing UI functionalities and improve interaction responsiveness.
2. **Hardware Communication Enhancements:**
    - Refactor the Arduino interface module to include error recovery, auto-detection of COM ports, and enhanced latency management.
    - Implement asynchronous logging for real-time error tracking.
3. **Standalone Packaging:**
    - Package the application using PyInstaller or cx_Freeze.
    - Test across multiple operating systems (Windows, macOS, Linux) to ensure compatibility.
4. **Extended Simulation Features:**
    - Integrate advanced data visualization libraries such as Plotly for interactive waveform graphs.
    - Add functionality for saving circuit configurations, loading previous sessions, and generating comprehensive experiment reports.
5. **User Feedback and Iteration:**
    - Implement a feedback module within the simulator to collect user input for continuous improvement.
    - Regularly update the documentation with new screenshots, sample experiment walkthroughs, and troubleshooting tips.

Forward-Looking Statement: The evolution of this tool is guided by both technical feasibility and user feedback. Future updates will address current limitations while exploring innovative features such as real-time collaboration and integration with IoT devices for remote digital logic experiments.

8. Conclusion

The **Digital Logic Lab Simulator** stands as a versatile and powerful educational platform that bridges the gap between theoretical digital logic concepts and practical, hands-on experimentation. Through its dual-mode simulation—virtual and hardware-based—it not only demystifies the inner workings of digital circuits but also challenges users to engage deeply with the subject matter through interactive experimentation.

By carefully following this documentation, users can expect a smooth setup process, a rich, interactive learning environment, and a transparent view of the underlying technical architecture. The detailed code annotations, comprehensive troubleshooting guides, and clearly structured workflow diagrams ensure that this tool remains accessible and continually adaptable for both classroom and individual learning environments.

Additional Consideration: For developers and advanced users, the documentation provides ample guidance on extending the simulator's capabilities. Future work includes transitioning to a fully offline mode, exploring advanced simulation techniques, and enhancing hardware integrations to create a seamless educational tool that evolves with emerging digital logic technologies.

Next Steps: As you venture into using the Digital Logic Lab Simulator, consider exploring the advanced topics provided in the annexure. Experiment with modifying the codebase, suggest improvements based on direct use, and participate in the community of users who are driving the next generation of interactive educational tools in digital logic design.