Functional Specification Document

▶ Contents □

Web Interface for FPGA Simulator

1. Introduction

1.1 Purpose

This document outlines the functional specifications for a web interface that visualizes signal propagation inside an FPGA (Field-Programmable Gate Array). The primary purpose of this software is educational, allowing students to observe and understand how signals propagate through an FPGA's components on a user-friendly web interface.

1.2 Project Scope

The project involves developing a web application that:

- Displays a 2D representation of an FPGA's layout after synthesis and Place & Route (P&R)
- Animates signal propagation through the FPGA's Basic Elements (BELs) and interconnections
- Allows users to control the simulation (next, previous)
- Provides navigation tools (zoom, pan) for exploring the FPGA layout
- Includes a backend system for teachers to upload Verilog and SDF files

1.3 Intended Audience

This specification is intended for:

- The development team responsible for implementing the web interface
- Teachers who will use the backend to create educational content
- Stakeholders reviewing the project requirements and deliverables
- Students who will interact with the frontend to see how signals flow in an FPGA

---

2. System Overview

2.1 System Architecture

The FPGA simulator web interface is designed as a multi-tier architecture, ensuring separation of concerns and scalability. The architecture comprises the following components:

2.1.1 Presentation Layer (Frontend)

- **Web Interface**: A user-friendly interface simulating signal propagation in an FPGA.
- **Visualization**: An animated flow of signals from the starting point to the ending point.
- **Interactive Controls**: Features like zoom, pan, next, and previous buttons for navigating and controlling simulations.

2.1.2 Application Layer (Backend)

- **Web Server**: Creates a link between the frontend and backend via an API.
- **Simulation Engine**: The simulations run based on the pivot file.
- **Business Logic**: Contains the core logic for processing uploaded files, managing simulations, and handling user interactions.

2.2 User Roles

2.2.1 Teacher

- **Responsibilities**: Upload Verilog applications and SDF files, manage application examples, and ensure the accuracy of simulation data.
- **Permissions**: Access to the backend interface for file uploads and data management. Ability to create and update application examples.
- **Interactions**: Primarily interacts with the backend to upload files and configure simulations.

2.2.2 Student

- **Responsibilities**: Run simulations and observe signal propagation within the FPGA.
- **Permissions**: Access to the frontend interface to control simulations and view results.
- **Interactions**: Interacts with the web interface to navigate the 2D FPGA layout, control simulations, and analyze results.

---

3. Functional Requirements(FR)

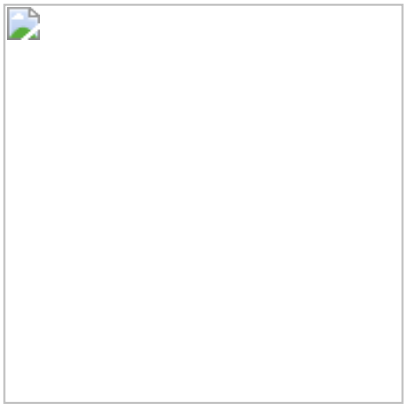
3.1 Frontend Web Interface (Student Use)

3.1.1 Application Selection

- **FR-1.1**: The interface will display a loaded application example.
- **FR-1.2**: Users shall be able to interact with the interface without any problems.

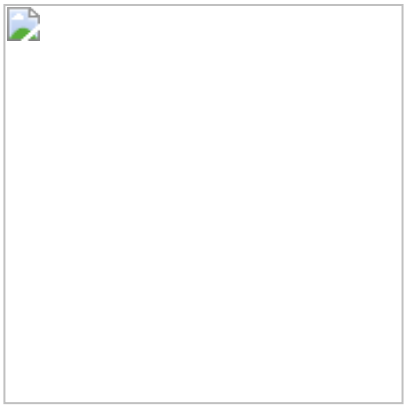
3.1.2 2D Visualization

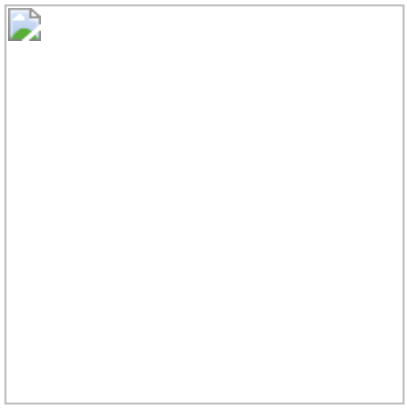
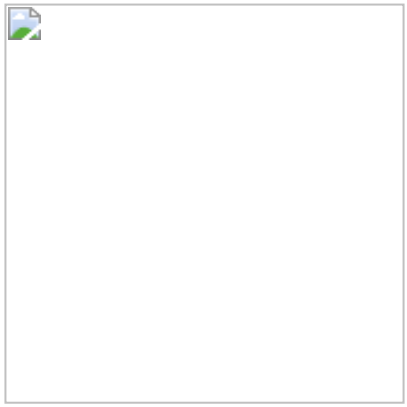
- **FR-2.1**: The interface shall display a 2D representation of the FPGA layout showing:
    - Basic Elements (BELs) used by the application
    - Signal routes between BELs
    - Current state of signals (active/inactive)
- **FR-2.2**: BELs shall be visually distinguishable by type (flip-flop, LUT).
- **FR-2.3**: Signal routes shall be visible and distinguishable from BELs.
    - BELs shall be displayed in block form, while signals shall be displayed in line form.
        - **Flip-flop**: Displayed in a horizontal rectangular form.
        - **LUT**: Displayed in a vertical rectangular form.
        - **Signals**: Displayed in vertical and horizontal lines.
- **FR-2.4**: Active signals shall be visually highlighted during simulation.
    - Inactive signals shall be displayed in gray (**color: #808080**).
    - Signal path from input to output shall be displayed in blue (**color: #0000FF**).
    - The current active signal state in the simulation shall be displayed in red with a layer shadow effect (**color: #FF0000**).



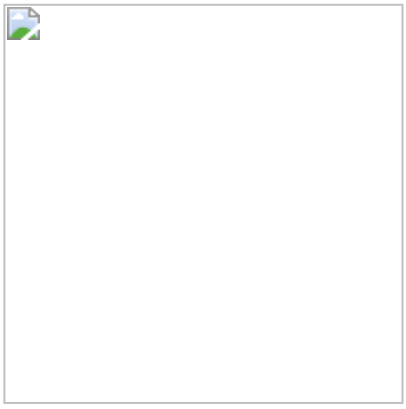
3.1.3 Navigation Controls

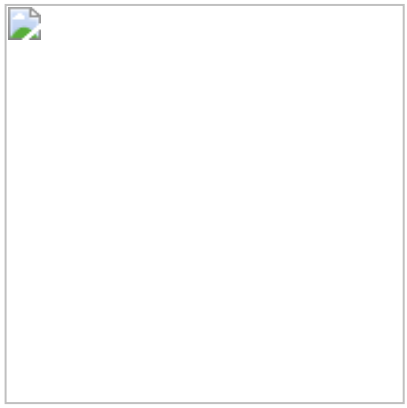
- **FR-3.1**: Users will be able to zoom in and out of the 2D view.
    - The zoom button with positive sign is for zoom in whereas the other one with negative sign is for zoom out

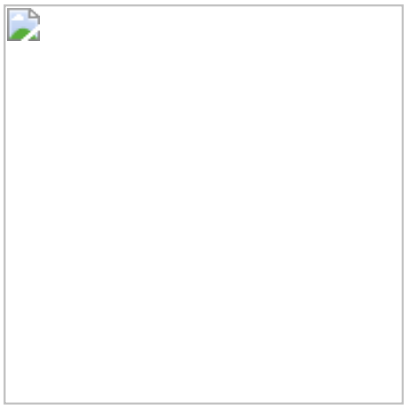


- **ZOOM-IN:** 
- **ZOOM-OUT:** 

- **FR-3.2**: Users shall be able to through the simulation process step by step around the 2D view.



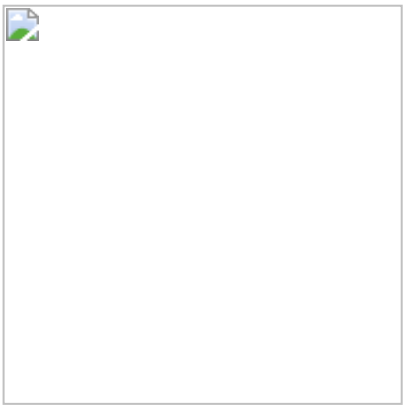
- **Steps:** Guide you to know precisely at what level you are in the simulation
- **Prev:** Helps you to see the previous step of the simulation 

- **Next:** Helps you to go to the next step of the simulation 

3.1.4 Simulation Controls

The simulation will be controlled manually; there will be no automatic display of the current flow in the simulation. The **next and previous** buttons will be used to navigate through the simulation.

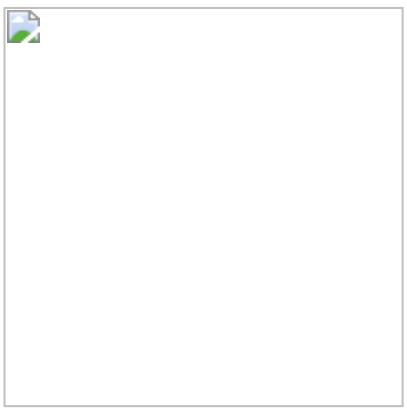
3.1.5 Information Display

- **FR-5.1**: The interface will display the current simulation time.
- **FR-5.3**: The interface shall provide information of what is happening at each step.
- **FR-5.4**: The interface may display a timeline of signal changes. 

3.2 Program Flow

Both the flowchart and pseudocode represent the flow of the simulation process on the web interface.

3.2.1 Flowchart



3.2.2 Pseudo-code

```
// Pseudo-code for FPGA Simulator with Zoom and Step Control

// Define inputs
Input2 = Input2
Input3 = Input3
Clock = Clock

// Define LUT and Flip-flop
LUT1 = CombinationalLogic(Input2, Input3)

Flipflop1 = DFlipFlop(Clock, LUT1)
Flipflop3 = DFlipFlop(Clock, Flipflop1)

// Initialize simulation state
CurrentStep = 0
ZoomLevel = 1  // 1 represents normal view, >1 represents zoomed-in

// Simulation steps
InitializeComponents()

While (SimulationRunning) {
  If (ButtonPressed) {
    Log("Button pressed")

    If (Button == "ZoomIn") {
        ZoomLevel += 1
    } Else If (Button == "ZoomOut") {
        If (ZoomLevel > 1) {
            ZoomLevel -= 1
        }
    } Else If (Button == "Next") {
        If (ClockEdgeDetected) {
            Log("Clock on")
            UpdateFlipFlops("Next")
            CurrentStep += 1
        }
    } Else If (Button == "Prev") {
        If (CurrentStep > 0) {
            UpdateFlipFlops("Prev")
            CurrentStep -= 1
        }
    }
  }
```

```
  Log("Output")
}

// Function Definitions
Function CombinationalLogic(inputs) {
  // Define the logic for LUTs based on inputs
  Return logic_output
}

Function DFlipFlop(Clock, Input) {
  If (ClockEdgeDetected) {
    Return Input
  }
  Return PreviousState
}

Function UpdateFlipFlops(direction) {
  If (direction == "Next") {
    Flipflop1.Update(LUT1)
    Flipflop3.Update(Flipflop1)
    Log("Flipflop1 updated")
    Log("Flipflop3 updated")
  } Else If (direction == "Prev") {
    // Revert flip-flop states to previous step
    RevertFlipFlopStates(CurrentStep)
    Log("Reverted to previous state")
  }
}

Function DisplayWithZoom(zoomLevel) {
  // Adjust display based on zoom level
  ApplyZoom(zoomLevel)
}

Function ApplyZoom(zoomLevel) {
  // Implement zoom functionality
  AdjustDisplayScale(zoomLevel)
}
```

 Note: This mockup represents the initial version of our simulation interface. Some modifications will be made during the implementation phase. Below are key points to consider:

3.3 Key Points for Implementation and Improvement.

- **Improve View Clarity:**
    - **Wire Organization**: Group wires horizontally and vertically to maintain a clean and organized layout.
    - **Element Simplification**: Remove elements like **Enable** and **Reset** to enhance clarity.
    - **Menu Icon**: Display a small menu icon at the top right of the screen. Clicking this icon will allow teachers to upload new Verilog and SDF files.
    - **First-Time User Notification**: If a user runs the simulation for the first time, display a small pop-up notification prompting them to upload their files to initiate the simulation.
- **Element Identification:**
    - **Grid Labels**: Place row and column numbers on the borders for easy element identification, avoiding clutter within the elements.
- **Refine Transition Animation:**
    - **Signal Propagation**: Enhance the transition animation to accurately depict signal propagation through the wires.
    - **Visual Effects**: Adjust the blur effect to better illustrate signal movement. Use a mix of red and green colors, each 1mm in length, on active signal lines to distinguish different signal states.
- **Distinguish Input/Output Elements:**
    - **Clear Categorization**: Treat input and output elements as distinct entities, similar to flip-flops and LUTs. Avoid mixing them with names like clock, enable, or reset to maintain clear categorization.
- **Comprehensive Log Information:**
    - **Detailed Logs**: Expand the log to include all clock edges and Basic Logic Elements (BLEs) information, indicating whether they are **on** or **off**. This provides a complete understanding of signal movement within the FPGA.

Improvements Made:

- **Clarity and Conciseness**: Simplified language and removed redundancies for better readability.
- **Structured Formatting**: Organized points into clear categories with sub-points for easier navigation.
- **Consistent Terminology**: Used consistent terms and removed unnecessary capitalization for a more professional tone.
- **User-Friendly Instructions**: Provided clear instructions for user interactions, such as the menu icon and first-time user notification.

These improvements should make the notes more accessible and actionable for the implementation team.

3.3 Backend System (Teacher Use)

3.3.1 Application Upload (Synthesis)

- **FR-1.1**: Teachers shall be able to upload Verilog application files through a secure and user-friendly interface.
- **FR-1.2**: The system shall support batch uploads of multiple Verilog files to streamline the process for teachers.
- **FR-1.3**: The system shall validate uploaded files for format correctness and provide immediate feedback on any errors or inconsistencies.
- **FR-1.4**: Upon successful validation, the system shall store the uploaded files securely in a designated storage solution.

3.3.2 Processing Pipeline (Place)

- **FR-2.1**: The system shall convert the uploaded Verilog and SDF files into a standardized JSON pivot file format accessible by the frontend.
- **FR-2.2**: The backend shall process the converted JSON files to generate the necessary simulation data, including netlist and timing information.
- **FR-2.3**: The backend shall dynamically calculate signal propagation timing in milliseconds each time the user interacts with the simulation control panel, ensuring real-time updates.
- **FR-2.4**: The processing pipeline shall be optimized for performance, handling multiple simulations concurrently without significant delays.

3.3.3 Pre-Visualization (Route)

- **FR-3.1**: The backend shall generate a 2D representation of the FPGA layout from the processed data, highlighting the different Basic Logic Elements (BLEs) used for signal propagation.
- **FR-3.2**: The visualization data shall be structured to support interactive features such as zoom and navigation within the frontend interface.
- **FR-3.3**: The backend shall ensure the visualization data is synchronized with the simulation state, providing an accurate representation of signal propagation.

3.3.4 Example Management

- **FR-4.1**: Teachers shall be able to upload both Verilog and SDF files for each application example.
- **FR-4.2**: The system shall maintain a catalog of application examples, allowing teachers to manage, update, or delete examples as needed.

3.3.5 Intermediary File Format

- **FR-5.1**: The system shall define an intermediary JSON pivot file format to bridge between backend processing and frontend visualization.
- **FR-5.2**: The JSON format shall include:
    - **Module Name**: The name of the Verilog module.
    - **Signal Information**: Names, types (input/output), and values of signals.
    - **Signal Propagation Timing Data**: Timing details for each signal path.
    - **Simulation Event Sequence**: Information about the clock signal and other relevant events.
- **FR-5.3**: The JSON format shall be versioned to accommodate future enhancements and ensure backward compatibility.

---

4. Use Cases

4.1 Use Case: Running a Simulation

Use Case ID	UC-01
Actors	Student
Preconditions	The teacher has preloaded an application example.
Description	The student runs the simulation.
Postconditions	The FPGA layout is visualized, showing signal propagation in blue color.
Alternative Flows	Student prev/next/zoom in/out the simulation mid-run.

4.2 Use Case: Uploading a Verilog Application

Use Case ID	UC-02
Actors	Teacher
Preconditions	The teacher has a Verilog file and SDF file.
Description	The teacher uploads the files, which are processed into a pivot file for visualization.
Postconditions	The application is available for students.
Alternative Flows	System rejects incompatible files and provides error messages.

5. Personas

5.1 Persona: Alex (The Student)

- **Role:** Engineering Student
- **Experience:** Basic digital electronics knowledge
- **Goals:**
  - Understand FPGA signal propagation
  - Visualize routing mechanisms
  - Practice with different examples
  - Self-paced learning
- **Pain Points:**
  - Abstract concepts difficult to grasp
  - Limited hands-on experience
  - Need for interactive learning tools

5.2 Persona: Dr. Smith (The Teacher)

- **Role:** FPGA Course Instructor
- **Experience:** 15+ years teaching digital electronics
- **Goals:**
    - Upload and manage FPGA examples
    - Create comprehensive learning materials
    - Monitor student progress
    - Demonstrate complex FPGA concepts
- **Pain Points:**
    - Time-consuming setup of examples
    - Difficulty in explaining dynamic concepts
    - Limited visual tools for demonstration

---

6. Non-Functional Requirements

6.1 Performance Requirements

- **NFR-1.1**: The web interface shall load within 3 seconds on a standard broadband connection to ensure a responsive user experience.
- **NFR-1.2**: The simulation animation shall run smoothly with no noticeable lag, maintaining at least 30 frames per second.
- **NFR-1.3**: The simulation running time between Basic Logic Elements (BLEs) should not match the timing specified in the SDF file, in order to ensure a clean visual signal propagation.
- **NFR-1.4**: The backend shall process file uploads and generate simulation data within 10 seconds for typical use cases.

6.2 Usability Requirements

- **NFR-2.1**: The interface shall be intuitive and require minimal training for students, with clear navigation and interactive elements.
- **NFR-2.2**: The interface must be compatible with major web browsers, including the latest versions of Chrome, Firefox, Safari, and Edge.
- **NFR-2.3**: The interface must be responsive and usable on devices with screens of at least 1024x768 resolution, including tablets and mobile devices.
- **NFR-2.4**: The simulation shall offer manual playback options, with user controls for previous, next step and adjusting the zoom and pan.
- **NFR-2.5**: Minimize the learning curve for new users, especially students, by providing tooltips, tutorials, and a comprehensive help section.

- **NFR-2.6**: The system shall ensure the web interface is accessible to users with disabilities, adhering to WCAG 2.1 Level AA accessibility standards.

6.3 Reliability Requirements

- **NFR-3.1**: The system shall maintain an uptime of at least 99.5%, excluding planned maintenance windows.
- **NFR-3.2**: The system shall implement robust error handling and recovery mechanisms to manage unexpected issues gracefully, with automated failover processes.
- **NFR-3.3**: The system shall provide informative and user-friendly error messages for common issues, guiding users on how to resolve them.
- **NFR-3.4**: Critical system components shall have redundancy to ensure high availability and minimize downtime.

6.4 Maintainability Requirements

- **NFR-5.1**: Ensure the codebase is well-documented, modular, and adheres to best practices for ease of maintenance and future development.
- **NFR-5.2**: Implement a process for regular updates and patches to address bugs, performance issues, and security vulnerabilities promptly.
- **NFR-5.3**: Use monitoring tools to track system performance, resource utilization, and user activity, identifying potential issues proactively.
- **NFR-5.4**: Maintain a version control system (e.g., Git) to manage code changes, facilitate collaboration, and enable rollbacks if necessary.
- **NFR-5.5**: Conduct regular code reviews and automated testing to ensure code quality and catch issues early in the development cycle.

---

7. Data Model

7.1 Entities and Relationships

- **Application**: Represents a Verilog application that can be simulated.
    - **Attributes**:
        - Application ID (Unique Identifier).
        - Name (Name of the application).
        - Description (Brief description of the application).
        - Verilog Code (The Verilog code for the application).
        - SDF Code (The SDF file content for timing information).
        - Upload Timestamp (Date and time when the application was uploaded).
- **Simulation**: Represents a simulation run of an application.
    - **Attributes**:
        - Simulation ID (Unique Identifier).
        - Application ID (Reference to the associated application).

        - Timestamp (Date and time when the simulation was initiated).
        - Status:
            - **Step:** Displaying in number, at which level you are on the simulation.
            - **Log:** Displaying detail information on what is happening at each step.

- **Netlist**: Represents the netlist generated from an application.

    - **Attributes**:
        - Netlist ID (Unique Identifier)
        - Application ID (Reference to the associated application)
        - Verilog Netlist (The netlist in Verilog format)
        - SDF File (The SDF file content)

7.2 Relationships

- A **Teacher** can upload multiple **Applications**, each consisting of Verilog and SDF files.
- A **Student** can run multiple **Simulations** on available **Applications**.
- Each **Simulation** is associated with one **Application** and generates a corresponding **Netlist**.
- **Users** can be either **Teachers** or **Students**, with different permissions and access levels.

7.3 Data Flow

- **Teacher**:

    - Uploads Verilog and SDF files to the backend.
    - The backend processes these files to generate a pivot file in JSON format, which includes necessary simulation data.
    - Selects an **Application** from the preloaded examples.

- **Student**:

    - Initiates a **Simulation**.
    - Views the simulation results in the web interface, with real-time updates on signal propagation.

7.4 Data Integrity and Validation

- **File Validation**: Implement validation rules to ensure that uploaded Verilog and SDF files are in the correct format and contain all necessary information.
- **Data Consistency**: Maintain consistent relationships between entities (e.g., a **Simulation** must be associated with a valid **Application** and **Netlist**).
- **Error Handling**: Implement error handling mechanisms to manage invalid data entries or processing failures, providing informative feedback to users.
- **Access Control**: Ensure that only authorized users can upload applications or initiate simulations, based on their role (Teacher or Student).

7.5 Example Data Model Diagram

- **Applications** (linked to Users)
    - **Attributes**: Application ID, Name, Description, Verilog Code, SDF Code, Upload Timestamp.
- **Simulations** (linked to Applications)
    - **Attributes**: Application ID, Timestamp, Status.
- **Netlists** (linked to Simulations)
    - **Attributes**: Netlist ID, Application ID, Verilog Netlist, SDF File, Generation Timestamp.

---

8. Technical Implementation Considerations

8.1 Frontend Technologies

8.1.1 HTML/CSS

- **Purpose:** Structure and style the web interface.
- **Frameworks:** Consider using CSS, no need of using any framework.

8.1.2 JavaScript

- **Purpose:** Add interactivity to the web interface.
- **Frameworks/Libraries:** React.js for building dynamic user interfaces.

8.1.3 WebGL or Canvas

- **Purpose:** Render the 2D FPGA layout and signal propagation visualizations.
- **Libraries**: D3.js for complex visualizations.

8.2 Backend Technologies

8.2.1 Programming Languages

- **Node.js:** JavaScript runtime for server-side development.

8.2.2 Web Frameworks

- **Express.js (Node.js):** For creating RESTful APIs.

8.3 Integration Points

8.3.1 Overview

The integration tools facilitate seamless interaction between the various components of the FPGA simulator web interface. They ensure that data flows smoothly from file uploads to simulation visualization, leveraging a pivot file format (JSON) to bridge the gap between backend processing and frontend display.

8.3.2 Key Integration Components

1. **File Upload and Conversion**:
    - **Verilog and SDF Files**: Teachers upload Verilog application files and corresponding SDF files via the web interface.
    - **Conversion to JSON**: Upon upload, these files are converted into a standardized JSON format. This JSON acts as a pivot file, containing all necessary data for simulation, including netlist and timing information.
    - **Tools**: Utilize backend services or scripts to automate the conversion process, ensuring consistency and accuracy in the JSON output.
2. **Backend Processing**:
    - **JSON Parsing**: The backend processes the JSON file to extract relevant data for simulation setup.
    - **Simulation Engine**: The backend simulation engine uses the parsed JSON data to configure and run the simulation.
3. **Frontend Visualization**:
    - **Data Retrieval**: The frontend retrieves simulation data from the backend via RESTful APIs, using the JSON format for data interchange.
    - **Visualization Tools**: Use libraries like D3.js or Three.js to render the 2D FPGA layout and signal propagation based on the JSON data.
4. **API Integration**:
    - **RESTful APIs**: Define clear and secure API endpoints for communication between the frontend and backend.
    - **Authentication**: Implement secure authentication mechanisms to protect API access and ensure data integrity.

8.3.3 Benefits

- **Seamless Data Flow**: Ensures smooth data transition between different system components, from file upload to simulation visualization.
- **Scalability**: Supports the addition of new features or tools without disrupting existing workflows.
- **Consistency**: The use of a standardized JSON format maintains data consistency across the system.
- **Real-time Interaction**: Enhances user experience with real-time updates and interactive visualizations.

---

9. Appendices

9.1 Glossary of Terms

- **FPGA (Field-Programmable Gate Array):** An integrated circuit designed to be configured by the customer or designer after manufacturing. It consists of basic elements and preconfigured electrical signal routes. The selected FPGA for this project is a NanoXplore NGultra (with VTR flow a basic Xilinx serie 7 model).
- **BEL (Basic Element):** The fundamental hardware electrical resources available inside the FPGA, such as flip-flops, Look-Up Tables (LUTs), and Block RAM.
- **Application:** The function or design intended to be executed within the FPGA, typically developed using Verilog.
- **Synthesis:** The process of translating the application's high-level description (e.g., Verilog code) into an electrical equivalent, resulting in a netlist. Tools like Impulse or Yosys (in VTR flow) are used for this process.
- **P&R (Place and Route):** The process of placing the netlist components within the FPGA's available BELs (Place) and selecting routes for signals between each BEL (Route). Tools like Impulse or VPR (for VTR flow) are used for this process.
- **Simulator:** A tool that compiles Verilog testbenches and applications, executing the simulation of signal behavior over time. Modelsim (or Icarus Verilog for VTR flow) is used for this purpose.
- **Software:** The web application developed as part of this project, designed to visualize and simulate signal propagation within an FPGA.
- **Verilog:** A hardware description language (HDL) used to model electronic systems. It is used to describe the behavior and structure of digital circuits.
- **SDF (Standard Delay Format):** A file format used to specify timing information for digital circuits, including delays and timing constraints.
- **Netlist:** A description of the connectivity of an electronic design, listing the components and their interconnections. It is generated during the synthesis process.
- **JSON (JavaScript Object Notation):** A lightweight data interchange format used to structure data in a human-readable way. In this project, JSON is used as an intermediary format to bridge backend processing and frontend visualization.
- **Web Interface:** The user-friendly frontend of the software, allowing students to visualize and interact with FPGA simulations.
- **Simulation Engine:** The backend component responsible for running simulations based on the pivot file and user interactions.
- **Backend:** The server-side part of the application, handling file uploads, processing, and managing simulation data.
- **Frontend:** The client-side part of the application, providing the user interface for interacting with the simulation.
- **Pivot File:** A standardized JSON file format used to bridge backend processing and frontend visualization, containing necessary simulation data.

- **VTR Flow:** A set of tools and processes used for FPGA design, including synthesis, place and route, and simulation. It is used as an alternative flow in this project.

9.2 References

- DigitalJS project: https://digitaljs.tilk.eu/ (https://digitaljs.tilk.eu/)
- Verilog to routing: https://github.com/verilog-to-routing/vtr-verilog-to-routing (https://github.com/verilog-to-routing/vtr-verilog-to-routing)
- OSSCAD project: https://github.com/YosysHQ/oss-cad-suite-build (https://github.com/YosysHQ/oss-cad-suite-build)
- YoWASP: https://yowasp.org/ (https://yowasp.org/)
- Python SDF-timing library: https://github.com/chipsalliance/f4pga-sdf-timing (https://github.com/chipsalliance/f4pga-sdf-timing)