

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

SCSE22-0964

**Full-stack Web Development for
Auto-Assessment Platform**

Final Report

Author

Chua Chong Yih (U2022784B)

Project Supervisor

Dr Loke Yuan Ren

Submitted in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Computer Engineering of the Nanyang Technological
University

School of Computer Science and Engineering
2023

Abstract

The field of Computer Science education has witnessed a surge in the number of students seeking to understand its intricate concepts. As the demand for quality education grows, the need for efficient evaluation tools becomes increasingly evident. While automated assessment platforms have proved invaluable in evaluating programming skills, they fall short when it comes to assessing Hardware Description Languages (HDLs). HDLs are specialized languages used to simulate digital circuits and systems, posing unique challenges that current assessment tools designed for General-Purpose Programming Languages (GPLs) cannot address.

This study builds upon the Automated Assessment Platform (AASP) developed by previous FYP works, enhancing its capabilities to accommodate HDL assessments. Leveraging the strengths of the existing AASP architecture and the scalable technology stacks, a comprehensive solution is devised to assess HDL assessments effectively. The approach is based on modifying the Judge0 framework, a recognized Online Judge System (OJS), to compile HDL code and visualize interactive waveforms generated by students' compiled HDL code.

Acknowledgement

I would like to express my sincere gratitude to Dr. Loke Yuan Ren for his invaluable guidance, support, and mentorship throughout the course of this project. I am deeply grateful for the opportunity to have worked under his supervision and for the invaluable lessons that I have learned during this journey.

Table of Contents

Abstract.....	i
Acknowledgement.....	ii
Table of Contents.....	iii
List of Figures.....	v
1 Introduction.....	1
1.1 Background.....	1
1.2 Prior Work.....	2
1.3 Problem Statement.....	2
1.4 Objectives.....	2
1.5 Scope.....	3
1.6 Organisation of Report.....	3
1.7 Project Plan.....	4
2 Literature Review.....	5
2.1 Related Works.....	5
2.1.1 HDLBits.....	5
2.1.1.1 Circuit Design Exercises.....	6
2.1.1.2 Verification: Reading Simulations.....	7
2.1.1.3 Verification: Writing Testbenches.....	8
2.1.1.4 Exercise Evaluation.....	9
2.2 HDL Assessments Technologies.....	10
2.2.1 Integration of Judge0.....	10
2.2.2 Waveform Visualization.....	11
2.2.2.1 GTKWave.....	11
2.2.2.2 EPWave.....	11
2.2.2.3 WaveDrom.....	12
2.2.2.4 VCDrom.....	13
3 Design Methodology.....	15
3.1 Use Case Diagram.....	15
3.2 Functional Requirements.....	16
3.2.1 HDL Assessment Integration.....	16
3.2.2 Component Validation.....	17
3.2.3 Compilation and Simulation.....	17
3.2.4 Waveform Visualization.....	17
3.3 Non-functional Requirements.....	18
3.4 Activity Diagrams.....	18
3.4.1 Create Question: Select Language Type.....	18

3.4.2 Create Question: Generate Module and Testbench Code.....	19
3.4.3 Create Question: Create HDL Test Case.....	20
3.4.2 Assessment Attempt.....	20
3.5 Design Frameworks.....	21
3.5.1 Wizard Design Pattern.....	21
3.5.2 Design Language System.....	21
3.6 Database Design.....	22
4 Implementation.....	23
4.1 System Architecture.....	23
4.2 Judge0 Modification.....	24
4.3 Waveform Visualization.....	25
4.3.1 VCDrom.....	25
4.3.2 WaveDrom.....	26
4.3.2.1 WaveDrom Integration.....	26
4.3.2.2 Mismatch Graph.....	26
4.4 Features.....	27
4.4.1 Language Selection.....	27
4.4.2 Code Snippets Creation.....	28
4.4.3 Module Generation.....	29
4.4.4 Testbench Generation.....	30
4.4.5 Test Case Creation.....	31
4.4.6 Question Attempt.....	32
4.4.7 Question Attempt: Module and Testbench Design.....	33
4.4.8 VCDrom.....	34
4.4.9 Waveform Comparison.....	35
5 Future Works.....	36
5.1 Additional HDL.....	36
5.2 More Question Types.....	36
5.3 Automatic Saving Code.....	36
5.4 Custom Judges.....	36
5.5 Unit Test.....	37
6 Conclusion.....	38
7 References.....	39

List of Figures

Figure 1 Gantt Chart.....	4
Figure 2 Combinational Logic Exercise of HDLBit.....	6
Figure 3 Sequential Logic Exercise of HDLBit.....	6
Figure 4 Reading Simulation Exercise of HDLBit.....	7
Figure 5 Writing Testbench Exercise of HDLBit.....	8
Figure 6 Test Case Results of HDLBit.....	9
Figure 7 Waveform drawn from GTKWave.....	11
Figure 8 Waveform drawn from EPWave.....	12
Figure 9 Textual description and timing diagram of WaveDrom.....	12
Figure 10 Timing diagram and shortcuts of VCDrom.....	13
Figure 11 AASP Use Case Diagram.....	14
Figure 12 Activity Diagram for Selecting Language.....	18
Figure 13 Activity Diagram for Generating Module and Testbench Code.....	18
Figure 14 Activity Diagram for Creating HDL Question.....	19
Figure 15 Activity Diagram for Attempting Assessment.....	19
Figure 16 Entity Relationship Diagram.....	21
Figure 17 System Architecture Diagram.....	22
Figure 18 Hardware Language Selection page.....	26
Figure 19 Code Snippet Creation page.....	27
Figure 20 Module Generation Modal.....	28
Figure 21 Module Design and Generated Testbench.....	29
Figure 22 Test Case Creation with Generated WaveDrom Output.....	30
Figure 23 Question Attempt page.....	31
Figure 24 Question Attempt page for Module and Testbench Design.....	32
Figure 25 VCDrom Modal.....	33
Figure 26 Test Case Detail Modal with Waveform Comparison.....	34

1 Introduction

1.1 Background

Programming assessments are crucial to evaluating a student's understanding of Computer Science (CS) concepts. With the increasing number of CS students and limited teaching staff, automated tools have become necessary to aid educators in code assessment [3]. However, existing code assessment tools cannot evaluate Hardware Description Languages (HDL). HDL is a specialized computer language used to simulate the behaviour of digital circuits and systems, which current assessment tools are not equipped to handle as they are developed for General-Purpose Programming Languages (GPL) such as Python and Java.

In contrast to traditional programming assessments for GPLs, which require students to write a single program to produce the expected output [2], HDL assessments involve multiple components, such as module design files, testbench files, and waveform dump files. These components can be tested in various combinations, such as requiring the student to write the testbench file to simulate the given waveform or the module design file to produce the output text.

Online Judge Systems (OJS), such as Judge0 [11], have been developed to accurately evaluate user-submitted algorithm source code in a standardized environment [1]. However, these systems only assess GPLs, which lack support for HDLs. Furthermore, compiling HDL code outputs a VCD file used to display a waveform, which is HDL-specific and unsupported by Judge0.

Automated assessment tools, such as LeetCode [12] and HackerRank [9], have become increasingly popular for conducting programming assessments in academic institutions. However, their subscription-based models make them economically unviable for many institutions.

1.2 Prior Work

The Automated Assessment Platform (AASP) was initially introduced by Soh Yan Quan, Kenneth as an in-house platform that supported code assessment and reporting. Subsequently, Yap Guan Sheng improved AASP by adding multiple-choice question assessments and an enhanced user interface (UI) [4]. However, this iteration of AASP contained critical bugs, security vulnerabilities, and maintainability issues [6]. Heng Fuwei, Esmond further improved AASP, focusing on security and analytics [15]. However, this iteration was developed using an architecture lacking proper security and scalability compared to the AASP introduced by Lee Jun Wei. Lee introduced a new version of AASP using scalable and maintainable technology stacks such as Judge0 [6]. The AASP was then further enhanced with proctoring features by Liu Wing Lam [7]. However, this latest version of AASP still lacks support for conducting HDL assessments.

1.3 Problem Statement

AASP currently lacks the capability to conduct assessments for HDLs, a specialized form of computer languages used to simulate digital circuits and systems, which involves intricate components such as module design files, testbench files, and waveform dump files. This absence of HDL assessment support poses a challenge for educators seeking a comprehensive evaluation solution within AASP. Addressing this gap is essential to ensure a comprehensive assessment experience that covers both GPLs and HDLs, enabling educators to accurately gauge student understanding of digital circuit behavior.

1.4 Objectives

The project will build and improve on the existing AASP developed by Lee to support HDL assessments. The project will also propose an effective approach for conducting automated assessments on HDL that considers different components

involved in HDLs and implement the proposed approach on the existing AASP developed by Lee and improve on it. The proposed approach leverages the scalability of AASP by modifying Judge0 to compile HDL and display an interactive waveform generated by the student's compiled code.

1.5 Scope

The scope of the project includes:

1. To study current existing HDL assessment platforms to identify desirable characteristics and areas of improvement.
2. To analyze and implement HDL tools that can aid in HDL assessments.
3. To plan, design and incorporate HDL assessments into AASP.

1.6 Organisation of Report

Chapter 1 Introduction: Provides an overview of the project's background, its objectives, and the scope of the study.

Chapter 2 Literature Review: Explores existing works and technologies related to HDL assessments.

Chapter 3 Design Methodology: Outlines the system's design requirements, accompanied by use case and activity diagrams.

Chapter 4 Implementation: Discusses the architecture and key features of the implemented system.

Chapter 5 Future Works: Explores potential areas for future development.

Chapter 6 Conclusion: Summarizes and provides a conclusion to the project.

1.7 Project Plan

Figure 1 shows the Gantt Chart outlining the timeline and key phases of the project. The project begins with the Discussion and Research phase, which aims to gather insights and background information crucial for project understanding. This phase is followed by Analysis and Planning, where comprehensive evaluation and strategic planning takes place to establish a strong foundation for the project's execution. The subsequent phases include a series of Implementations, including HDL Integration, VCDrom Modification, WaveDrom Integration, and HDL Question Type Integration. Each of these implementation phases focuses on incorporating and integrating specific functionalities or components crucial to the project's core objectives. As the project nears completion, the focus shifts towards the Deployment and Final Preparations phase, where the project is deployed to the school's server. Finally, the project concludes with the Documentation phase.

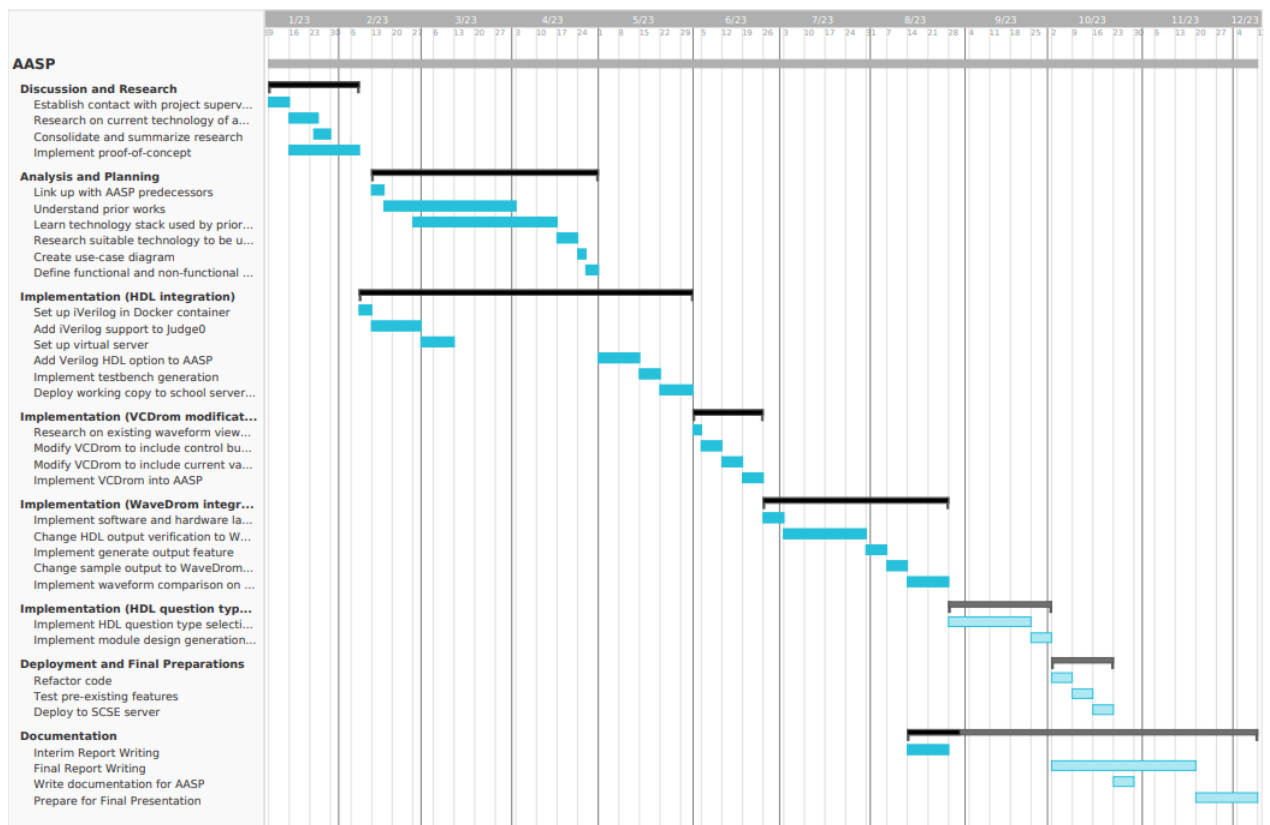


Figure 1 Gantt Chart

2 Literature Review

2.1 Related Works

Existing literature highlights the significance of automated assessment platforms in evaluating students' programming skills [1]. Traditional platforms like LeetCode and HackerRank have gained popularity due to their ability to assess GPLs. However, their subscription-based models render them economically unviable for many institutions. Moreover, these platforms primarily focus on GPLs, neglecting HDL assessments, which demand a different approach due to the complexity of digital circuit simulations and assessments.

2.1.1 HDLBits

HDLBits [10] centers its focus on offering a collection of small yet highly instructive circuit design exercises, primarily intended for teaching digital hardware design skills using Verilog HDL. While HDLBits may not possess the extensive functionalities commonly found in conventional programming assessment platforms, its role in promoting a comprehensive understanding of the methodologies and intricacies involved in Verilog HDL can be studied to better understand how HDL assessments can be conducted and incorporated into AASP.

The exercises available on HDLBits are organized into distinct categories, including circuit design exercises and verification-based exercises. The former challenges users to design digital circuits, developing their ability in constructing hardware components using Verilog HDL. On the other hand, the verification-based exercises include tasks such as reading simulations and creating testbenches—an important practice for ensuring the accuracy and functionality of HDL circuit designs.

2.1.1.1 Circuit Design Exercises

The screenshot shows the HDLBits website interface for the 'Mux2to1' exercise. The top navigation bar includes links for 'Problem Set', 'Simulation', 'My Profile', 'Help', and '01xz.net'. The left sidebar lists various circuit design topics, with 'Combinational Logic' and 'Multiplexers' expanded. The main content area for 'Mux2to1' includes a description of the task, a hint section, and a code editor for writing the Verilog solution. The module declaration is shown as follows:

```
module top_module(  
    input a, b, sel,  
    output out );
```

Figure 2 Combinational Logic Exercise of HDLBit

The screenshot shows the HDLBits website interface for the 'Count15' exercise. The top navigation bar includes links for 'Problem Set', 'Simulation', 'My Profile', 'Help', and '01xz.net'. The left sidebar lists various circuit design topics, with 'Sequential Logic' and 'Counters' expanded. The main content area for 'Count15' includes a description of the task, a timing diagram for the clock and reset signals, a hint section, and a code editor for writing the Verilog solution. The module declaration is shown as follows:

```
module top_module (  
    input clk,  
    input reset, // Synchronous active-high reset  
    output [3:0] q);
```

Figure 3 Sequential Logic Exercise of HDLBit

Circuit design exercises can be broadly categorized into two main groups: combinational logic (Figure 2) and sequential logic designs (Figure 3). Combinational logic exercises are relatively straightforward as they do not involve the complexities introduced by timing elements like clocks. This enables learners to grasp foundational concepts without the complexity of timing considerations. In

these exercises, the output depends solely on the present input, evaluating the understanding of logic gates, multiplexers, decoders, and similar fundamental components.

On the other hand, sequential logic exercises delve into more complex concepts by incorporating timing elements. This opens up the possibility of designing circuits with memory elements and components that exhibit changing behavior over time. A noteworthy feature of these exercises is the incorporation of waveform diagrams, which provide visual representations of expected output behaviors. These visual aids allow learners to comprehend the change of signals over time, thus aiding in the comprehension of how clock-driven circuits function.

2.1.1.2 Verification: Reading Simulations

The screenshot displays the HDLBits website interface. The top navigation bar includes links for 'HDLBits', 'Problem Set', 'Simulation', 'My Profile', 'Help', and '01xz.net'. A search bar is located on the right. The left sidebar contains a list of exercises categorized under 'Getting Started', 'Verilog Language', 'Circuits', and 'Verification: Reading Simulations'. The 'Sequential circuit 7' exercise is selected. The main content area shows the title 'Sim/circuit7' and a description: 'This is a sequential circuit. Read the simulation waveforms to determine what the circuit does, then implement it.' Below the description is a waveform diagram with inputs 'clk' and 'a', and output 'q'. The 'clk' signal is a periodic square wave. The 'a' signal is a single pulse. The 'q' signal is a square wave that changes state at each clock edge when 'a' is high. Below the waveform is a 'Module Declaration' section with the following code:

```
module top_module (  
    input clk,  
    input a,  
    output q );
```

Below the module declaration is a section titled 'Write your solution here' with a 'Load' button and a text area for the solution.

Figure 4 Reading Simulation Exercise of HDLBit

Figure 4 shows another category of HDL learning within HDLBits, which involves verification exercises, particularly those centered around reading simulation waveforms. This category of exercises tests learners' proficiency in comprehending simulation results and subsequently replicating corresponding module designs.

In these exercises, learners are presented with simulation waveforms that represent the behavior of a specific module. The objective is for learners to interpret the waveforms, deciphering the underlying logic and functionality. By doing so, they demonstrate their ability to understand and analyze the dynamic behavior of digital circuits. Subsequently, learners are tasked with recreating an equivalent module design that produces the observed waveform patterns. This practical approach engages learners in reverse-engineering the logic necessary to generate the desired output, thereby reinforcing their comprehension of circuit behavior and design principles.

2.1.1.3 Verification: Writing Testbenches

The screenshot shows the HDLBits website interface. The top navigation bar includes links for 'HDLBits', 'Problem Set', 'Simulation', 'My Profile', 'Help', and '01xz.net'. A search bar is located on the right. The left sidebar contains a list of topics: 'Getting Started', 'Verilog Language', 'Circuits', 'Verification: Reading Simulations', and 'Verification: Writing Testbenches'. Under 'Verification: Writing Testbenches', there is a sub-menu with 'Clock', 'Testbench1', 'AND gate', 'Testbench2', 'T flip-flop', and 'CS450'. The main content area is titled 'Tb/clock' and shows a simulation waveform for a clock signal 'clk' with a period of 10 ps. Below the waveform, there is a 'Module Declaration' section with the code 'module top_module ();'. A 'Write your solution here' section contains a text area with a 'Load' button and a 'Load a previous submission' dropdown. The text area contains the code 'module top_module ();', 'endmodule', and a line number '4'.

Figure 5 Writing Testbench Exercise of HDLBit

Writing testbenches (Figure 5) involves creating testing code that generates specific inputs for a module, monitors the outputs, and checks whether they align with the expected results. This practice not only evaluates learners' grasp of HDL concepts but also tests their capability to construct comprehensive testing scenarios. Learners must exhibit a deep understanding of the module's functionality to design test cases that thoroughly exercise the module's capabilities. The process encourages critical thinking and meticulous design considerations, as learners must envision various scenarios and edge cases to ensure comprehensive testing coverage.

2.1.1.4 Exercise Evaluation

Effectively evaluating student-generated code and presenting assessment outcomes in an informative manner is critical for enhancing learning comprehension. Distinct from assessments involving GPLs, the evaluation of solutions in HDL assessments often necessitates more complex techniques due to the unique characteristics of HDLs. While GPL assessments can generally rely on simple output comparisons, HDL assessments, particularly those involving sequential logic, require a more detailed approach due to the waveform nature of their output.

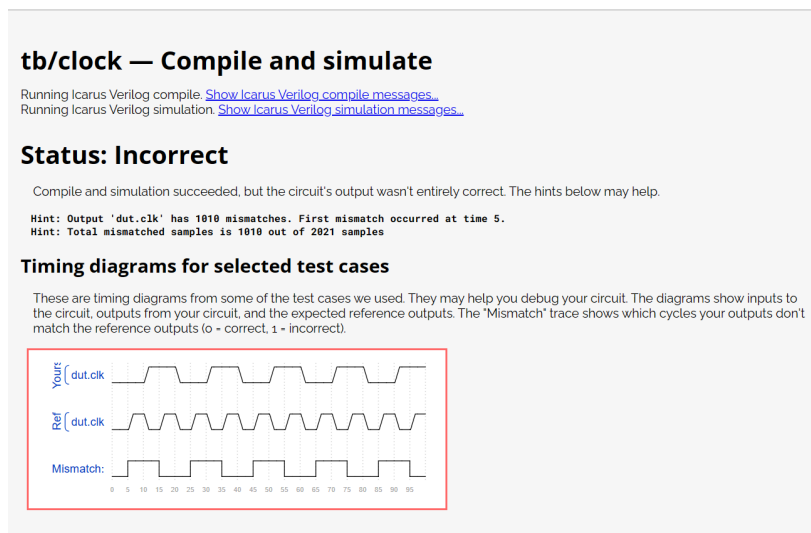


Figure 6 Test Case Results of HDLBit

HDLBits tackles this challenge with a distinctive approach to assessment outcome presentation (Figure 6). Unlike simple output comparisons, waveform-based assessments require a comparison between the expected output waveform and the student solution waveform. HDLBits employs an insightful visualization technique that compares these two waveforms side by side, highlighting points of divergence with a mismatch graph. This approach provides students with a better understanding of where their solution deviates from expectations in terms of timing and signal behavior.

By offering a comprehensive visualization of the discrepancy between expected and actual waveforms, students are able to pinpoint exactly where their solution goes

wrong. HDLBits' approach to presenting assessment outcomes significantly contributes to the learning process, ensuring that students not only identify errors but also grasp the underlying reasons, thereby enhancing their overall proficiency in HDL-based circuit design.

2.2 HDL Assessments Technologies

This section explores a suite of technologies aimed at enhancing the integration of HDL assessment in AASP. By offering an array of efficient tools and methods, this collection of technologies aims to significantly enhance the process of creating and attempting HDL assessment, fostering a more engaging, insightful, and efficient learning journey.

2.2.1 Integration of Judge0

At the core of the AASP lies the utilization of Judge0, an online code execution engine. For AASP to effectively include HDLs and extend its capabilities to encompass HDL assessments, Judge0 must gain the capability to compile HDL code. However, HDL is not currently featured in the official list of supported languages within Judge0's repertoire. Therefore, the task of incorporating HDL as a supported language requires a deeper exploration of Judge0's inner workings to facilitate this expansion.

Judge0's fundamental architecture stems from a compiler image that houses an array of compilers catering to various programming languages. This array empowers Judge0 to compile code across its spectrum of supported languages. This architecture can be visualized from the system architecture diagram in Section 4.1. By integrating the required tools and compilers for HDL languages, Judge0 can be equipped to compile HDL-based submissions. Furthermore, this approach's versatility extends to the potential addition of other languages in the future, ensuring that Judge0's functionality remains scalable.

2.2.2 Waveform Visualization

The integration of HDL into AASP places waveform visualization at the forefront. This is especially significant for sequential circuits, where HDL outputs in the form of Value Change Dump (VCD) file which can be visualized as timing diagrams. Waveforms emerge as essential tools for enhancing the comprehension of compiled codes. This section delves into a comprehensive exploration of waveform visualization tools available in the market. This exploration spans the spectrum from static to interactive visualization options, all tailored to suit the unique requirements of AASP's HDL integration.

2.2.2.1 GTKWave

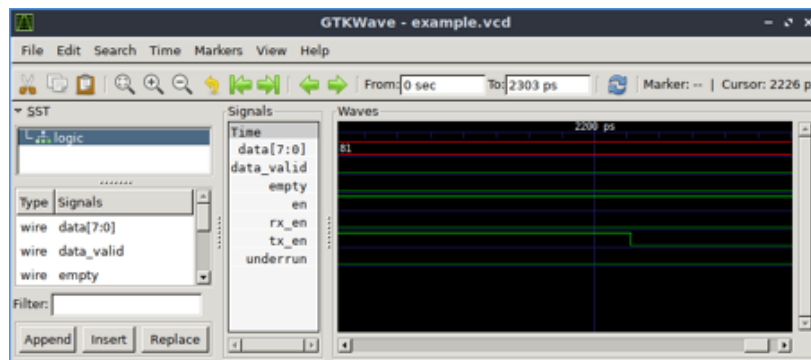


Figure 7 Waveform drawn from GTKWave

GTKWave (Figure 7) is a standalone open-source wave viewer program [8] for VCD files featured by Icarus Verilog, equipped with its own user interface for rendering waveforms. In the context of AASP's integration, however, seamless incorporation of GTKWave would demand a mechanism for real-time streaming, a task that would be infeasible due to the constraints of latency. The nature of GTKWave, designed as a standalone program with its own user interface, makes direct integration complex and impractical within the existing AASP framework.

2.2.2.2 EPWave

EDA Playground Wave (EPWave) [5] is a free open source interactive browser-based wave viewer that supports VCD. While direct integration with AASP remains infeasible due to its non-open-source nature, the features and functionalities of EPWave serves as a valuable point of study to incorporate into AASP's wave viewer.

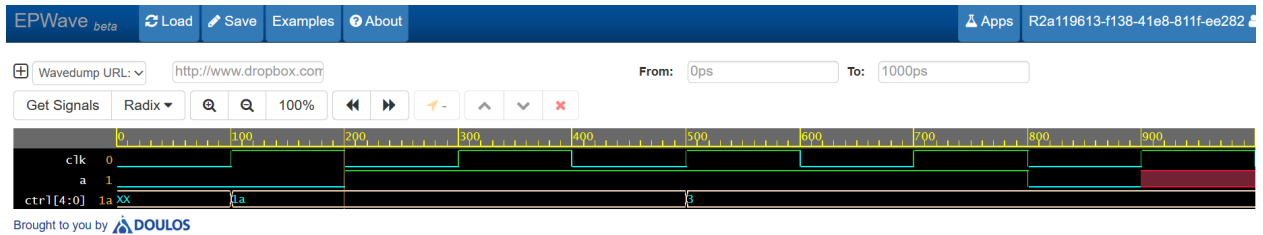


Figure 8 Waveform drawn from EPWave

EPWave has an intuitive user interface, supported by an array of user-friendly control buttons (Figure 8). This comprehensive toolkit allows users to navigate through their output waveforms, including the ability to zoom in and out, reset zoom settings, and scroll both left and right. This approach to waveform exploration caters to users' need for precision, clarity, and effortless interaction.

2.2.2.3 WaveDrom

WaveDrom [13] is a tool that renders timing diagrams or waveforms through simple textual description (Figure 9). This tool is notably employed by HDLBits, which will make any potential integration of HDLBit's exercise evaluation feature simpler.

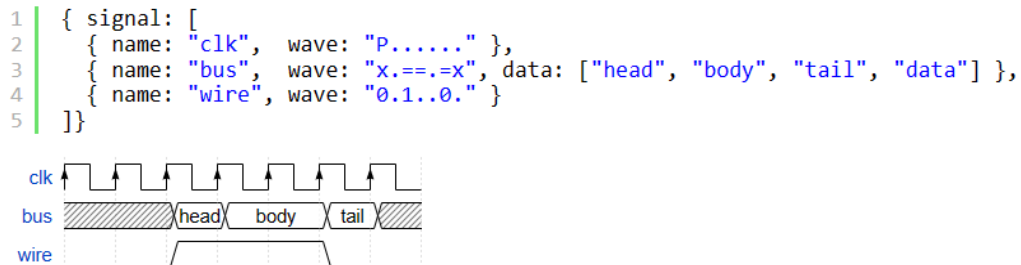


Figure 9 Textual description and timing diagram of WaveDrom

In the context of AASP, WaveDrom offers a powerful solution for waveform comparison. Its description language can be harnessed to represent expected output, facilitating precise comparison against student answers. This approach stands in contrast to VCD verification, which introduces additional meta-data and potential complexities, along with hard to understand outputs. By harnessing WaveDrom's simple description language, AASP gains an efficient mechanism for waveform validation, further enhancing its capabilities in HDL assessments.

2.2.2.4 VCDrom

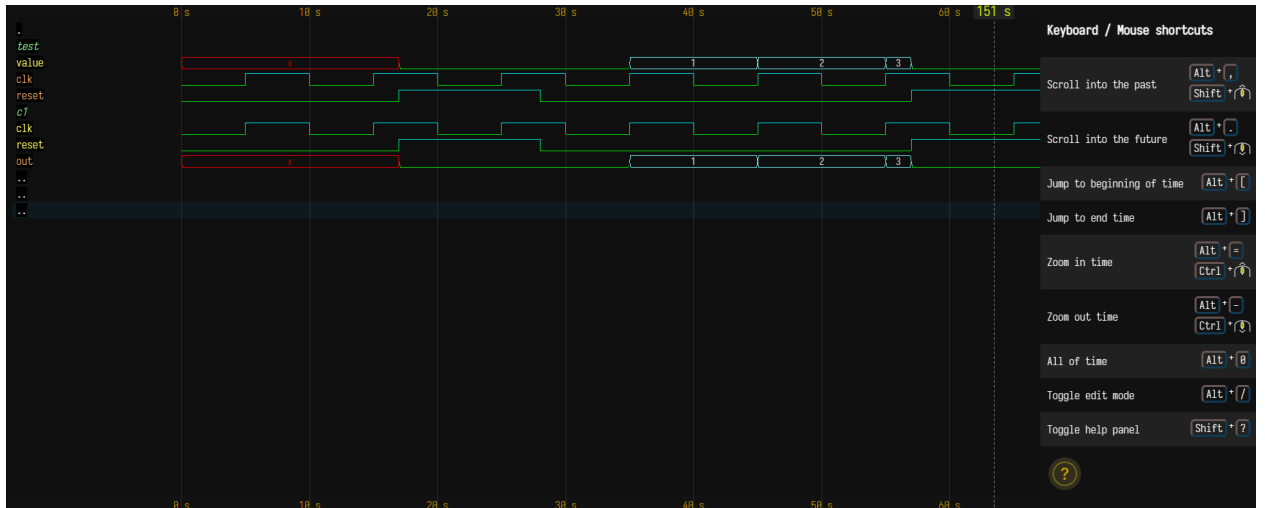


Figure 10 Timing diagram and shortcuts of VCDrom

VCDrom [14] is a standalone VCD viewer using Progressive Web App (PWA), developed by WaveDrom (Figure 10). This interactive waveform viewer serves as a complement to the WaveDrom's static waveform rendering capabilities.

However, while VCDrom presents valuable interactive attributes, it falls short in certain aspects for seamless integration with AASP. For instance, VCDrom employs keyboard shortcuts to navigate through its waveform interface, a feature that might be less intuitive for first-time users. To optimize the user experience, a more accessible solution involving a straightforward array of control buttons for each of these shortcuts could be explored.

3 Design Methodology

3.1 Use Case Diagram

Figure 11 is the use case diagram of AASP which illustrates the various interactions and possible scenarios between users and AASP.

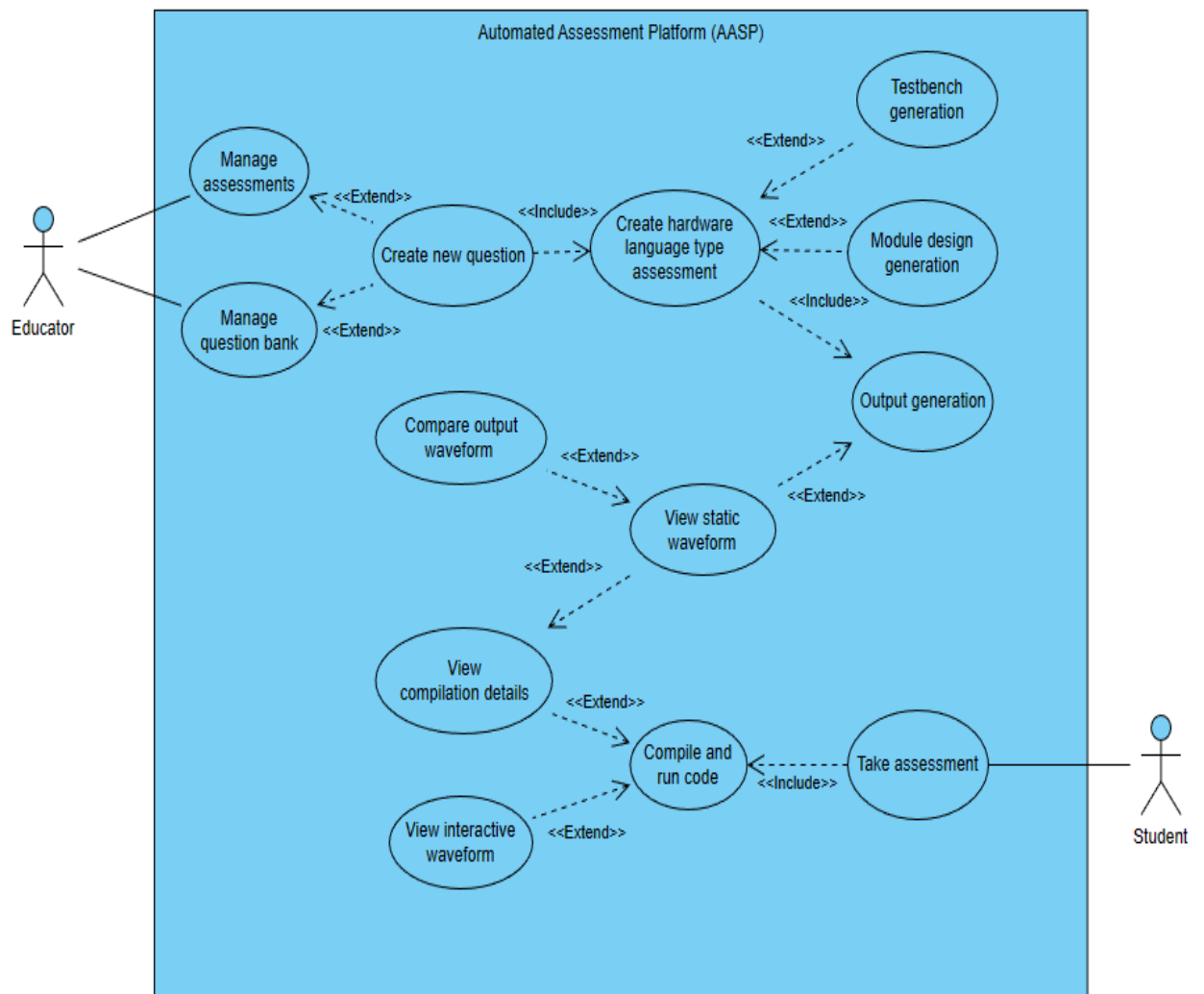


Figure 11 AASP Use Case Diagram

3.2 Functional Requirements

3.2.1 HDL Assessment Integration

1. AASP must be extended to support HDL-based assessments by incorporating a dedicated module for handling HDL components.
2. Educators must be able to create, edit, and manage HDL assignments through an intuitive user interface.
3. The system must support different assessment configurations.
 - 3.1. Educators must be able to configure testbench based HDL questions.
 - 3.2. Educators must be able to configure module design based HDL questions.
 - 3.3. Educators must be able to configure module-design and testbench based questions.
4. The system must support the generation of HDL boilerplate code.
 - 4.1. Educators must be able to generate module design boilerplate code through an interactive user interface.
 - 4.1.1. Educators must be able to specify input ports and the number of bits.
 - 4.1.2. Educators must be able to specify output ports and the number of bits.
 - 4.2. Educators must be able to generate testbench boilerplate code based on a given module design code.
5. System must be able to generate an output waveform by providing module design and testbench code.
 - 5.1. Educators must be able to specify the generated output waveform as the expected answer for a given test case.
 - 5.2. Educators must be able to generate an output waveform on the test case creation page.
 - 5.3. Students must be able to generate an output waveform on the code question attempt page.

3.2.2 Component Validation

1. The system must be able to validate the correctness of uploaded module design and testbench files.
 - 1.1. The system must be able to verify the correctness of expected output by comparing with student generated waveform data.
2. Comprehensive error messages must be provided to students for quick identification and resolution of issues in their submissions.

3.2.3 Compilation and Simulation

1. The modified Judge0 framework must be integrated into AASP to handle HDL compilation and simulation.
2. Students' HDL code must be compiled and simulated against provided module design or testbench files, generating output waveforms for visualization.

3.2.4 Waveform Visualization

1. The system must display interactive waveform visualizations to students, enabling them to analyze the behavior of their HDL code during simulation.
2. The system must display the expected output as a static waveform.
 - 2.1. The system must display the expected output waveform on the question attempt screen.
 - 2.2. The system must display the expected output waveform on the test case detail modal screen.
 - 2.2.1. The system must compare the students' compiled waveform with the expected waveform by grouping them together.
 - 2.2.2. The system must display a mismatch graph to compare the outputs of the two waveforms.
 - 2.2.3. The mismatch graph must display any mismatch between the two waveforms with a high, and any match with a low.

3.3 Non-functional Requirements

1. The modified AASP shall be compatible with a variety of web browsers and devices, ensuring accessibility for a diverse user base.
2. The platform shall maintain high availability, ensuring uninterrupted access for users during critical assessment periods.
3. HDL simulations and waveform visualizations shall provide real-time or near-real-time feedback to students to enhance their learning experience.
4. Response times for uploading, compiling, and simulating HDL code shall be optimized for efficiency.

3.4 Activity Diagrams

Detailed diagrams illustrating the flow of existing features can be found in the works of previous final year project reports [6, 7]. These diagrams provide insights into the newly added HDL assessment functionalities of AASP. In this section, the focus shifts to a set of activity diagrams that specifically showcase the integration of HDL into AASP. These diagrams offer a step-by-step walkthrough of the processes involved in enhancing AASP to support HDL assessments, outlining user actions, system responses, and data flow.

3.4.1 Create Question: Select Language Type

Creating questions within AASP requires educators to specify the language type – hardware or software. The process for selecting software language adheres to the framework outlined by Lee [6]. On the other hand, choosing hardware language allows educators to select the desired HDL question type, including Module Design, Testbench Design, and Module and Testbench Design (Figure 12).

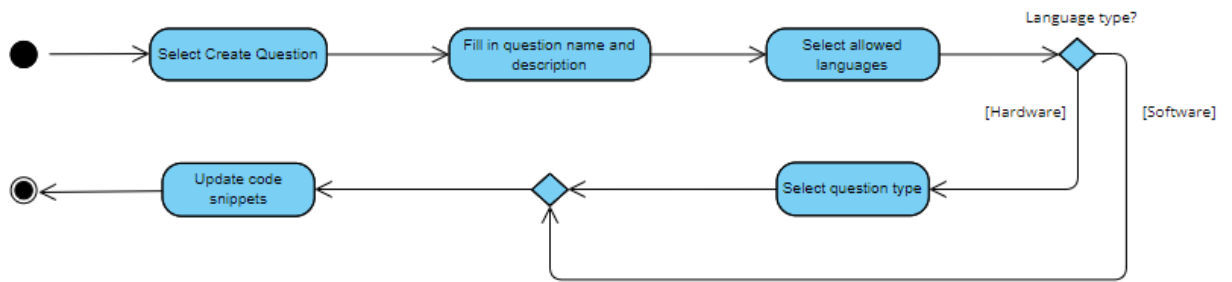


Figure 12 Activity Diagram for Selecting Language

3.4.2 Create Question: Generate Module and Testbench Code

On the create test case page, a section for educators to input their solution code is provided. This section allows educators to generate module code through a popup modal form that requires them to key in their desired input and output ports, which will generate a template module code with all ports declared. They can also generate a template testbench based on the provided module code, along with an apply to all option to update all test cases with the testbench code in the solution section (Figure 13).

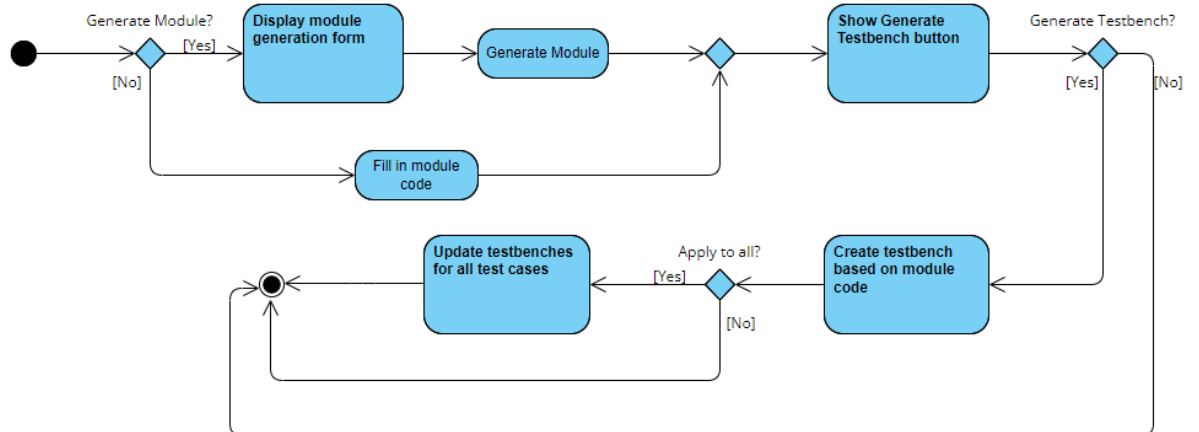


Figure 13 Activity Diagram for Generating Module and Testbench Code

3.4.3 Create Question: Create HDL Test Case

Educators are required to input a module or testbench solution, which will be compiled against their test case code to generate the expected test case output, which is then visually represented as a waveform. The educator can also choose to add additional internal test cases and repeat the process (Figure 14).

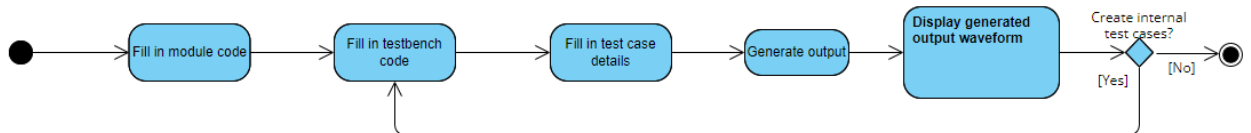


Figure 14 Activity Diagram for Creating HDL Question

3.4.2 Assessment Attempt

Users can explore their compiled code through an interactive waveform when attempting a HDL question. Upon selecting the option for more detailed insight into compiled or submitted attempts, a comparison waveform will be displayed, along with a mismatch graph (Figure 15).

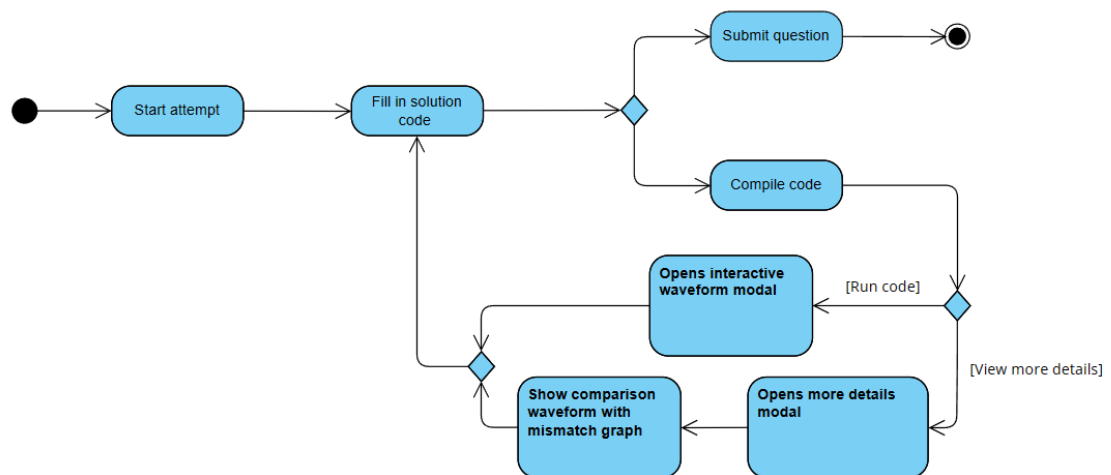


Figure 15 Activity Diagram for Attempting Assessment

3.5 Design Frameworks

This section provides insights into the UI design frameworks utilized within AASP, highlighting key considerations when designing new features for AASP.

3.5.1 Wizard Design Pattern

The Wizard Design Pattern is characterized by a sequential, step-by-step approach that guides users through complex tasks or processes. Each step is presented to the user in a clear and structured manner, with informative descriptions and user-friendly navigation.

In particular, AASP employs this design pattern in the question creation process, which comprises multiple sequential steps. This process begins by requesting the user to provide a description for the question. The user is then prompted to select the programming language of choice. Following this, the user is presented with options to specify the question type. Lastly, the user is required to input the test case details.

Each step in this process is deliberately designed to be simple and focused on a single action, thus minimizing user confusion. To further guide the user effectively, certain buttons, like the ‘Generate Testbench’ feature, become accessible only when the user updates the module code. This intuitive flow offers users implicit instructions, enabling them to navigate the system seamlessly and create questions with ease, while promoting a user-centric design approach.

3.5.2 Design Language System

The Design Language System within AASP serves as a comprehensive set of design guidelines, principles, and visual elements that maintain consistency and coherence throughout the user interface. Specifically, given that this version of AASP is a direct evolution of the system established by Lee [6], adhering to his design specifications is crucial to seamlessly integrate new features into the existing system architecture.

Bootstrap, a widely utilized CSS framework, played a central role in Lee's design of AASP. In this iteration of AASP, the continued use of Bootstrap for all design components maintains the system's familiar and user-friendly interface.

3.6 Database Design

Figure 15 is the updated Entity Relationship Diagram (ERD) of AASP's database schema which includes the necessary relationships for HDL integration, in addition to the design elements introduced in previous works [6, 7].

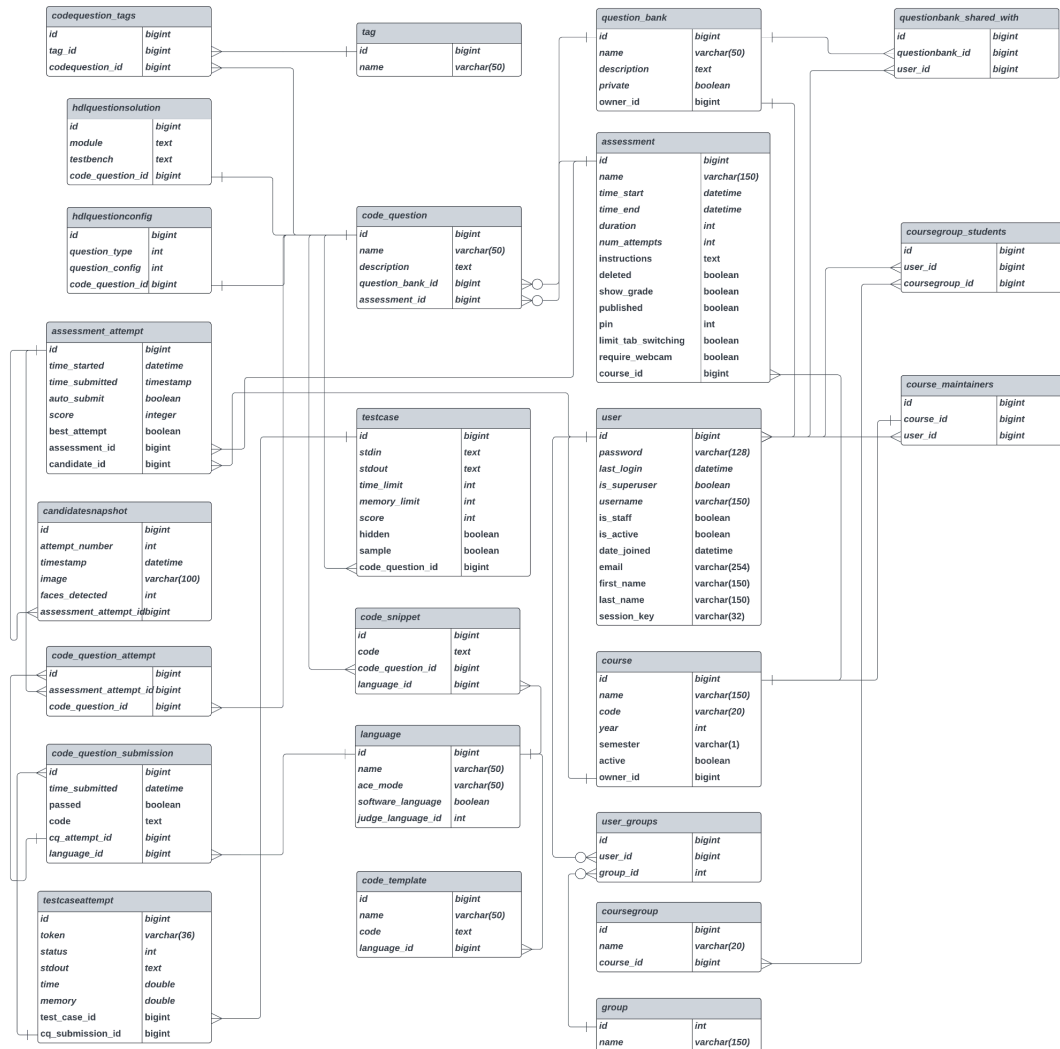


Figure 16 Entity Relationship Diagram

4 Implementation

4.1 System Architecture

Figure 16 is the system architecture diagram of AASP. Detailed descriptions of AASP's containerization strategy can be found in [6], and descriptions of AASP's proctoring features can be found in [7]. The following subsections focuses on the implementation of HDL related features in AASP.

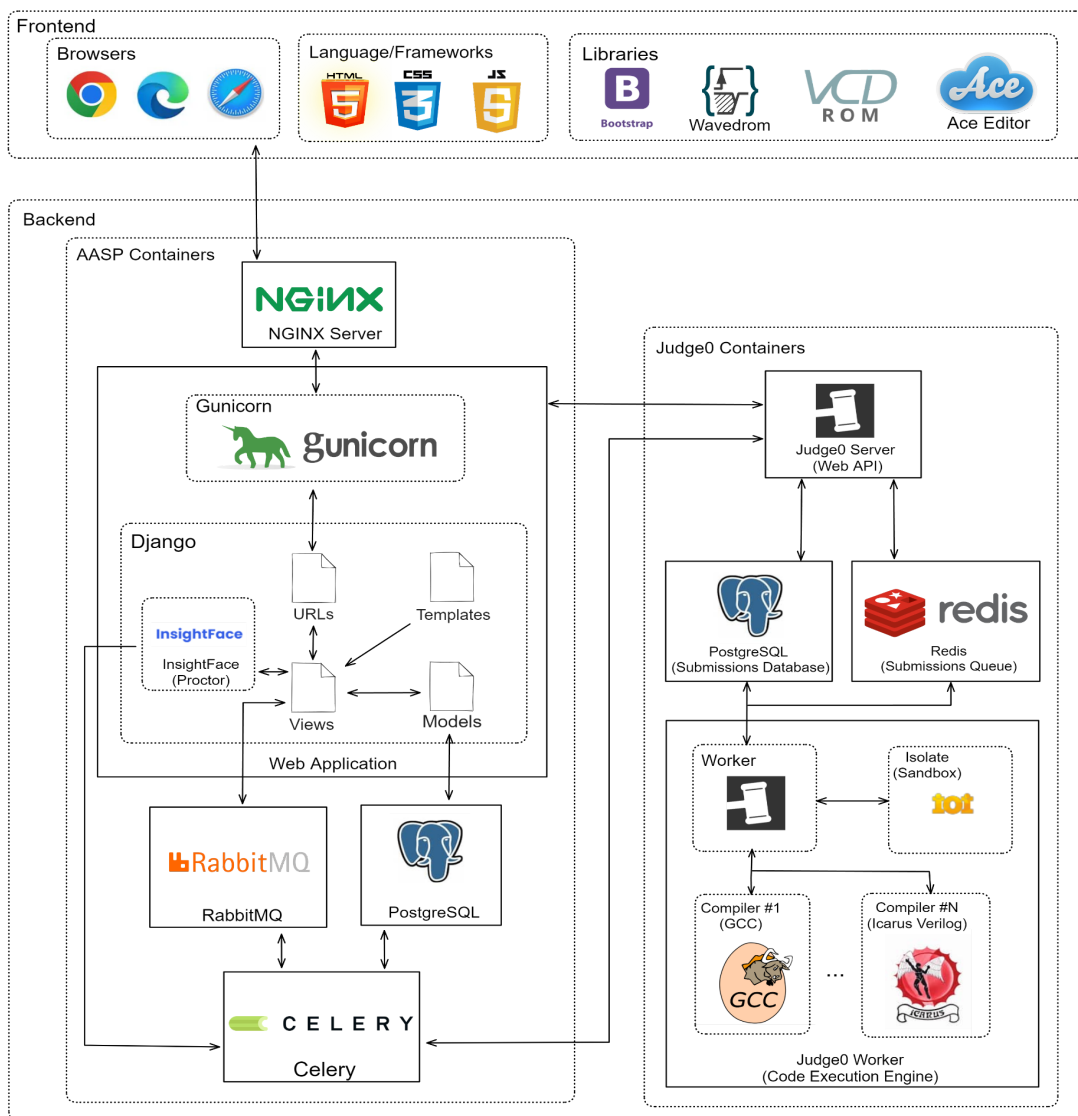


Figure 17 System Architecture Diagram

4.2 Judge0 Modification

Incorporating HDL support into Judge0 involves modifying its core compiler Docker image. The compiler image used by previous iterations of AASP contain an array of compilers with 46 languages, which is unnecessary for AASP's specific use case catered to NTU's curriculum. To streamline it, a more focused compiler image is curated by selecting only the compilers relevant to AASP, along with Icarus Verilog (iVerilog) to compile Verilog HDL code.

HDL, such as Verilog, typically involves multiple files and steps for compilation. For instance, Verilog involves a module design file and a testbench file. The compilation process starts with compiling the module design and testbench file. A runtime engine is then used to interpret the compilation output, which in turn generates a Value Change Dump (VCD) file.

Judge0 offers comprehensive support for multi-file programs, enabling users to define their compilation and execution scripts, thereby tailoring the platform to AASP's specific needs. To achieve this customization, a zip file containing the compile and run scripts is included in the POST request

However, using VCD as the output for comparison is suboptimal due to embedded metadata and limited readability for educators. To address this, WaveDrom's textual description format is adopted. This format provides a clearer representation of the output and enables the frontend to render the output waveform using WaveDrom's rendering engine. A script is integrated into the compiler image to convert the generated VCD file into WaveDrom format.

Despite not being directly utilized for output comparison, the VCD file remains essential for displaying the interactive waveform through VCDrom. However, the parameters returned by the base Judge0 do not support this functionality out of the box. As a result, additional modifications are made to introduce an extra parameter named "vcd_output". This parameter ensures that the VCD file is included in the

response, enabling WaveDrom to generate the interactive waveform representation on the frontend.

4.3 Waveform Visualization

4.3.1 VCDrom

In the context of adapting VCDrom to suit AASP's requirements, modifications were implemented to enhance the user experience (UX). The original VCDrom featured a landing page that necessitated user-uploaded VCD files for visualization, a workflow misaligned with AASP's needs. To address this, the landing page was removed, and AASP was reconfigured to facilitate a direct POST request with the VCD file as one of the parameters.

Given VCDrom's inherent lack of certain UX features, it underwent enhancements as detailed in Chapter 2. A primary focus was on incorporating control buttons, which provide users with more intuitive navigation. To achieve this, individual buttons were linked to specific keyboard actions. For example, the "Shift Right" control button was mapped to the keyboard shortcut "Alt + .". By clicking the "Shift Right" button, the corresponding keyboard action is effectively executed.

Incorporating the ability to select a specific time instance was a critical enhancement to address UX limitations. This modification significantly improved the clarity of signal output by associating the current value directly alongside each signal. To facilitate this, a new "selectvalue" layer was introduced into the VCDrom Document Object Model (DOM). This layer consists of a row of values positioned next to each port. Each time a user clicks on the waveform, the values displayed within the "selectvalue" layer are updated. These values are computed by identifying the selected time and determining the most recent value preceding that time point.

4.3.2 WaveDrom

4.3.2.1 WaveDrom Integration

The integration of WaveDrom into AASP required the conversion of VCD files into WaveJSON format for waveform visualization. Since test case validation is done by Judge0, this conversion had to be performed on the server side, during either compilation or execution within Judge0.

To perform this conversion, a dedicated script was developed and incorporated into Judge0's compiler image. During the submission to Judge0, the zip file containing execution instructions included the command to execute the script. This output was then considered as the standard output for comparison.

It is important to note that the execution of this conversion script adds an additional runtime overhead, which has the potential to trigger false negatives, particularly if the memory limit is exceeded. Consequently, it is necessary to account for this additional processing time when setting the runtime and memory limits for each test case. To address this concern, the default memory limit has been appropriately revised to accommodate the additional processing requirements.

4.3.2.2 Mismatch Graph

The construction of the mismatch graph required the grouping of input and output signals into distinct groups. However, a challenge arose as the WaveJSON format does not inherently specify whether signals are inputs or outputs. In order to address this issue, the suffixes '_in' and '_out' are appended to the respective signal names, clearly marking them as either input or output, prior to sending the data to Judge0. This added identifier enabled the frontend to distinguish between input and output signals.

Within the WaveJSON format, the use of a period ('.') signifies the extension of the previous state for an additional clock period. While this is an efficient representation,

it introduced an unnecessary complexity during comparison. Therefore, a two-step process was adopted to simplify this. Initially, both the compiled and expected WaveJSON output signals were modified to reflect the previous state for every occurrence of a period (‘.’). Subsequently, a straightforward comparison was done for the output signals at each clock period. A mismatch was recorded as ‘High’, while a match was recorded as ‘Low’ in the mismatch graph.

4.4 Features

4.4.1 Language Selection

Dedicated tabs on the language selection page differentiates between software and hardware assessments (Figure 18). The software assessment tab integrates the design pattern introduced by Lee, enabling users to select multiple programming languages.

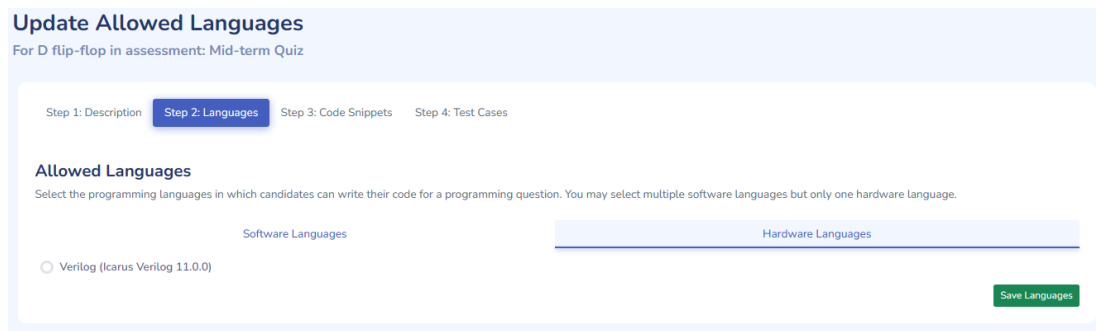


Figure 18 Hardware Language Selection page

On the other hand, hardware language assessment allows only a single hardware language to be selected for each question. This design consideration is due to the distinct nature of different hardware languages and recognizes that a single testbench, for instance, will not be universally applicable across different hardware languages due to syntax variations.

4.4.2 Code Snippets Creation

Update Code Snippets
For D flip-flop in assessment: Mid-term Quiz

Step 1: Description Step 2: Languages **Step 3: Code Snippets** Step 4: Test Cases

Code Snippets
Enter the default code snippets for each selected language. These default snippets will serve as the template that students will see when attempting the question.

Verilog (Icarus Verilog 11.0.0) - Module Verilog (Icarus Verilog 11.0.0) - Testbench

```
module main(clk, reset, out);  
    input clk, reset;  
    output out;  
  
    // enter your solution here...  
  
endmodule
```

Save Code Snippets

Figure 19 Code Snippet Creation page

The code snippets feature allows educators to define and customize the initial code templates that students use when attempting the code question (Figure 19). This serves as the foundation for students, providing them with a starting point and a structured format to follow while solving the given problem. In particular, the use of code snippets in HDL questions enables educators to define the input and output ports explicitly, which minimizes potential errors that might not be part of the intended testing scope.

4.4.3 Module Generation

Generate Module Code [X]

Define Module

Define a module and specify I/O Ports to generate a module template. For each port specified, MSB and LSB values will be ignored unless its Bus column is checked.

+

Module Name

counter

Port Name	Direction	Bus	MSB	LSB	Action
clk	input	<input type="checkbox"/>	0	0	
reset	input	<input type="checkbox"/>	0	0	
out	output	<input checked="" type="checkbox"/>	7	0	

Close Generate

Figure 20 Module Generation Modal

The module generation form provides a simple interface to declare the module name, and different number of ports (Figure 20). Each port must be declared with their port name, the direction as well as whether it is a bus signal, and if it is, the MSB and LSB has to be declared. The generated module will have the module and ports declared. Along with that, a testbench will be generated based on the module code.

4.4.4 Testbench Generation

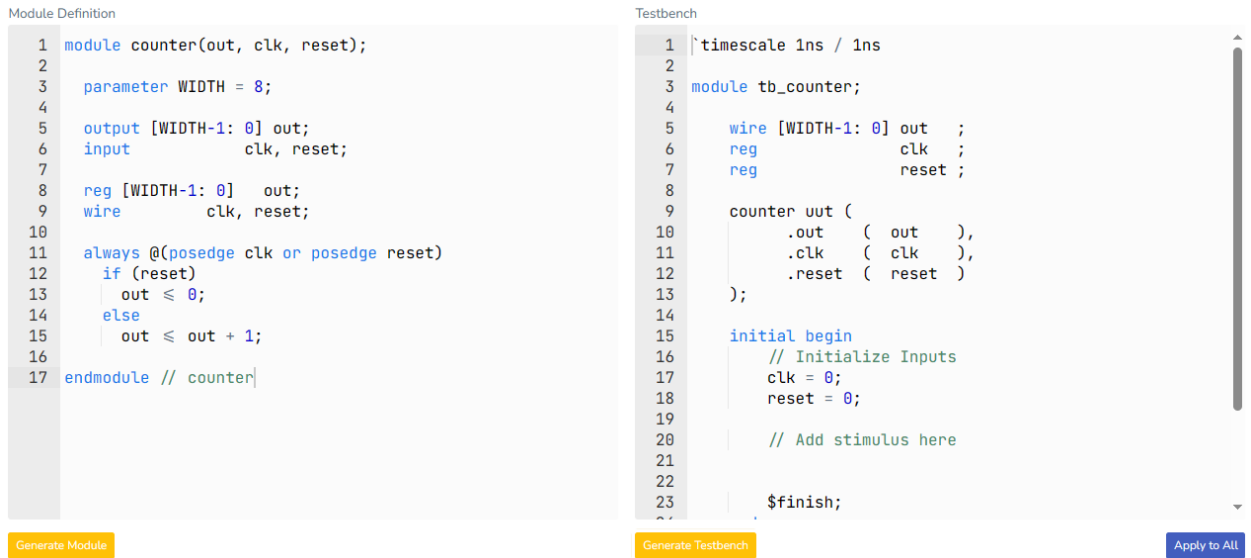


Figure 21 Module Design and Generated Testbench

The testbench generation feature within AASP simplifies the question creation process by transforming module design code into a boilerplate testbench code (Figure 21). The transformation includes declaring registers and wires for input and output ports, instantiating the top module, and generating clock and reset signals. An 'Apply to All' button will apply the testbench in the solution to all testbenches in the test case rows.

4.4.5 Test Case Creation

Sample Test Case

Internal Test Case 1

Internal Test Case 2

Sample Test Case

Testbench

```
1 module main_tb;
2
3   reg clk, reset;
4   wire out;
5
6   main uut (.clk(clk), .reset(reset), .out(out));
7
8   initial begin
9       // enter your solution here...
10
11
12   end
13
14 endmodule
```

Time Limit (s)

5

Memory Limit (KB)

40960

Score

0

Expected Output

Generate Output

Ready

Add Test Case

Save and Continue

Load from file

Figure 22 Test Case Creation with Generated WaveDrom Output

HDL assessments require educators to specify an output in WaveDrom format, as the expected output for a HDL question. AASP provides a "Generate Output" button which enables educators to compile the provided testbench and module design code inputs, subsequently converting them into WaveDrom format (Figure 22). The resulting waveform, derived from this compilation, is displayed below, allowing educators to visualize the expected output easily.

4.4.6 Question Attempt

The screenshot displays the Nanyang Technological University (NTU) Question Attempt interface. On the left is a navigation sidebar with links to Home, Courses, Enrol Students, Question Banks, Change Password, and Log Out. The main content area is titled "8 bit counter" and includes a description: "Implement an 8-bit counter module in Verilog. The module, named counter, should be able to count from 0 to 255 (in binary) and then reset back to 0." Below this is a "Port Declaration" section listing: out: 8-bit counter output, clk: clock input, and reset: reset input. A "Sample Input" section shows a Verilog testbench code snippet. Below the code is a "Sample Waveform" showing the clk and reset signals, and the output out[7:0]. On the right, the "Your answer" section shows the student's Verilog code for the counter module. The code is as follows:

```
1 module counter(out, clk, reset);
2
3     parameter WIDTH = 8;
4
5     output [WIDTH-1: 0] out;
6     input      clk, reset;
7
8     reg [WIDTH-1: 0] out;
9     wire      clk, reset;
10
11     always @(posedge clk or posedge reset)
12     if (reset)
13         out ≤ 0;
14     else
15         out ≤ out + 1;
16
17 endmodule // counter
```

Below the code are buttons for "Complete", "Run", and "Submit". To the right of these buttons, it says "Accepted" with a green checkmark and a link to "More Details". Below the code area is a "Submission History" section.

Figure 23 Question Attempt page

Beneath the question details of the sample question, the expected output waveform is displayed. Upon the successful compilation of the student's code, a "Run" button becomes accessible. Clicking this button opens a modal of VCDrom, displaying the waveform generated from the most recent successful compilation. In cases of compilation failure, the system will display the waveform from the last successful compilation (Figure 23).

4.4.7 Question Attempt: Module and Testbench Design

D flip-flop

Implement a D flip-flop in Verilog, with the provided code serving as a solution reference. The flip-flop should function based on the positive edge of the clock signal. It should also include an asynchronous reset functionality that sets the output to 0 when the reset signal is high.

Inputs:

- **clk** (1-bit signal) - Represents the clock input.
- **reset** (1-bit signal) - Represents the asynchronous reset input.
- **d** (1-bit signal) - Represents the data input.

Outputs:

- **q** (1-bit signal) - Represents the output of the D flip-flop.
- **qb** (1-bit signal) - Represents the complement of the output (i.e., the inverse of q).

Sample Waveform

Tags: Easy Verilog Double

Your answer

Language: Verilog (Icarus Verilog 11.0.0) Theme: Light (Cloud9 Day) [Reset]

```
1 // Design
2 // D flip-flop
3 module dff (clk, reset,
4   d, q, qb);
5   input      clk;
6   input      reset;
7   input      d;
8   output     q;
9   output     qb;
10
11   reg        q;
12
13   assign qb = ~q;
14
15   always @(posedge clk or posedge reset)
16   begin
17     if (reset) begin
18       // Asynchronous reset when reset goes high
19       q <= 1'b0;
20     end else begin
21       // Assign D to Q on positive clock edge
22       q <= d;
23     end
24   end
```

Accepted [More Details]

Submission History

Figure 24 Question Attempt page for Module and Testbench Design

Question attempt for a Module and Testbench Design question type is slightly different from other question types. This form of question type requires students to submit a solution for both module and testbench. A tab to switch between their module and testbench code is provided, and the compilation will be done with both their solutions (Figure 24).

4.4.8 VCDrom



Figure 25 VCDrom Modal

Upon selecting the "Run" option, a modal window will appear, allowing students to view the compiled code waveform using VCDrom. The modal (Figure 25) offers a range of control buttons, including shifting and zooming, that allow students to navigate through their compiled code effectively, enhancing their ability to analyze and understand the waveform representation within the platform.

4.4.9 Waveform Comparison

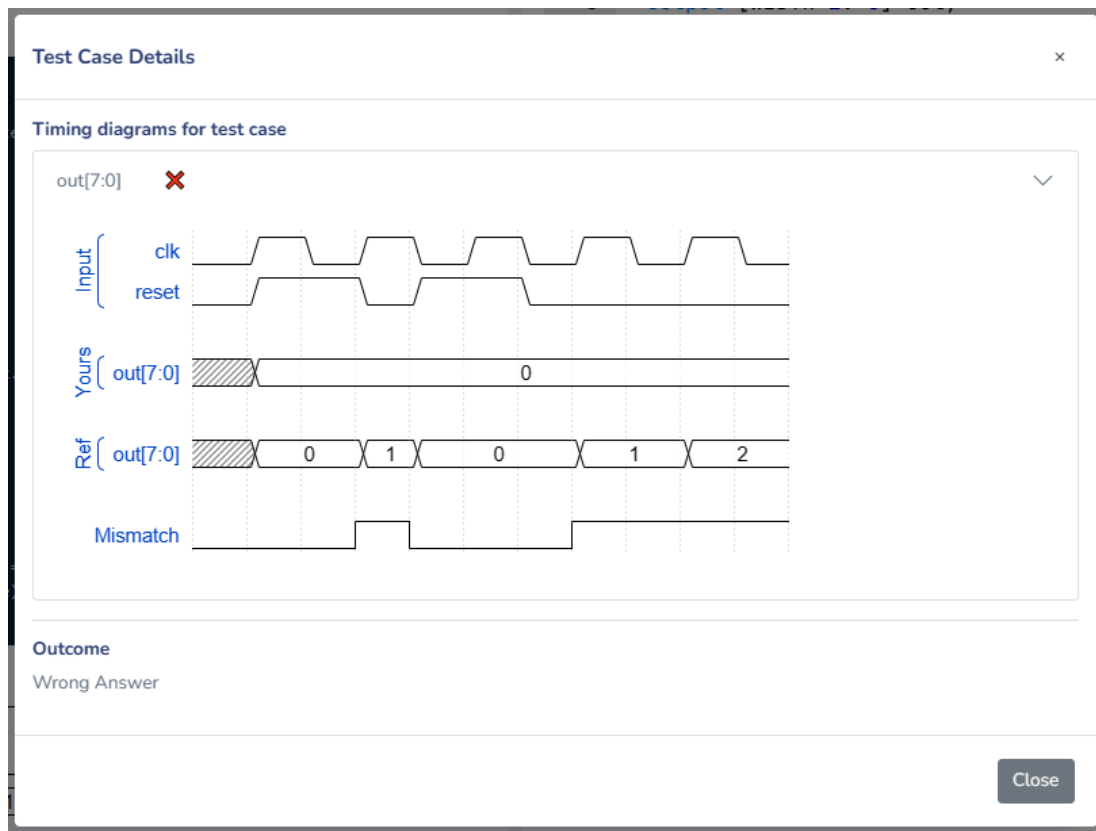


Figure 26 Test Case Detail Modal with Waveform Comparison

Upon selecting the "More Details" button, a modal window will appear, providing students with an expanded and comprehensive overview of their compiled solution. This modal (Figure 26) includes a waveform comparison display that groups input signals, juxtaposing the expected and student outputs. Below, a mismatch graph visually represents output discrepancies at each time instance, offering a thorough analysis of the student's code within HDL assessments.

5 Future Works

5.1 Additional HDL

The current framework within AASP only supports Verilog for its assessments. Although the current curriculum at NTU primarily focuses on Verilog, the inclusion of other HDLs, such as VHSIC Hardware Description Language (VHDL), can significantly enhance the future-proofing of AASP. Implementing support for additional HDLs, which can follow a similar implementation as detailed in Chapter 4, would involve a comprehensive understanding of the syntax and intricacies of the specific HDL.

5.2 More Question Types

Currently, AASP is exclusively tailored for programming questions. To expand its assessment capabilities and accommodate a broader spectrum of learning styles and content, additional question types can be introduced. These additions can include Multiple-Choice Questions (MCQ), short-answer questions. Other more advanced question types can include questions that enable users to respond with images, and questions presented in image format.

5.3 Automatic Saving Code

The integration of an automatic code-saving feature within AASP can significantly enhance user experience and workflow efficiency. By implementing a mechanism that automatically saves the student's code at regular intervals or upon specific user actions, the platform can prevent data loss and minimize the risk of unsaved progress.

5.4 Custom Judges

AASP, relying on the Judge0 platform, currently faces constraints in its ability to comprehensively access test cases. Judge0 employs a basic exact string comparison

on the expected output, limiting the system's capability to conduct detailed assessments, such as assigning scores to individual correct states within WaveJSON.

To overcome this limitation and introduce a more sophisticated evaluation mechanism, modifications to the Judge0 platform or the development of a custom judge can be done. Implementing a customized judge system could allow for the incorporation of tailored evaluation criteria, enabling the assessment of complex signal processing scenarios with greater precision and flexibility.

5.5 Unit Test

Due to the collaborative nature of the AASP project and its multiple iterations involving various contributors, the possibility of misinterpretation of certain functionalities cannot be overlooked. The introduction of a set of comprehensive unit tests can help ensure the accuracy and reliability of the system's components. Integrating unit testing within a Continuous Integration/Continuous Deployment (CI/CD) pipeline can further enhance the software development process, allowing for automated testing and deployment, thereby promoting a robust and error-free system architecture.

6 Conclusion

This report marks the completion of the development and implementation of a comprehensive full-stack web platform designed for the automated assessment of HDLs. The project focused on enhancing the existing AASP to effectively support and facilitate HDL assessments, essential for the simulation and evaluation of digital circuits and systems.

A key aspect of the project involved the adaptation of the Judge0 framework, an online code execution engine, to enable the compilation of HDL code and the visualization of interactive waveforms through the integration of WaveDrom and VCDrom tools. Additionally, the platform was equipped with essential features, including module and testbench generation, waveform comparison through a mismatch graph, and waveform visualization.

The report provides an overview of relevant works and technologies in the realm of HDL assessments by reviewing prominent tools and platforms. It also outlines the system design requirements and methodology, complemented by detailed use case scenarios and activity diagrams, offering a comprehensive understanding of the project's architecture and key functionalities.

For the future progression of the project, the report suggests potential development, including the expansion of support for additional HDLs, the introduction of diverse question types, the implementation of automatic code-saving functionalities, the development of custom judges, and the incorporation of comprehensive unit testing.

7 References

- [1] WasikSzymon, AntczakMaciej, BaduraJan, LaskowskiArtur, and SternalTomasz, "A Survey on Online Judge Systems and Their Applications," *ACM Computing Surveys (CSUR)*, Jan. 2018, doi: [10.1145/3143560](https://doi.org/10.1145/3143560).
- [2] C. Daly and J. Waldron, "Assessing the assessment of programming ability," Proceedings of the 35th SIGCSE technical symposium on Computer science education - SIGCSE '04, 2004, doi: [10.1145/971300.971375](https://doi.org/10.1145/971300.971375).
- [3] S. Combéfis, "Automated Code Assessment for Education: Review, Classification and Perspectives on Techniques and Tools," *Software*, vol. 1, no. 1, Art. no. 1, Mar. 2022, doi: [10.3390/software1010002](https://doi.org/10.3390/software1010002).
- [4] G. S. Yap, "Development of auto-assessment system," 2021, Accessed: Feb. 23, 2023. [Online]. Available: <https://dr.ntu.edu.sg/handle/10356/153233>
- [5] "EPWave Documentation," *EPWave*. Accessed: Feb. 27, 2023. [Online]. Available: <https://epwave.readthedocs.io>
- [6] J. W. Lee, "Full-stack web development for auto-assessment platform," 2022, Accessed: Feb. 23, 2023. [Online]. Available: <https://dr.ntu.edu.sg/handle/10356/162927>
- [7] W. L. Liu, "Full-stack web development for auto-assessment platform," 2023, Accessed: Aug. 27, 2023. [Online]. Available: <https://dr.ntu.edu.sg/handle/10356/165941>
- [8] "GTKWave," *GTKWave*. Accessed: Aug. 27, 2023. [Online]. Available: <https://gtkwave.sourceforge.net/>
- [9] "HackerRank - Online Coding Tests and Technical Interviews," *HackerRank*. Accessed: Feb. 23, 2023. [Online]. Available: <https://www.hackerrank.com/>
- [10] "HDLBits," *HDLBits*. Accessed: Feb. 27, 2023. [Online]. Available: https://hdlbits.01xz.net/wiki/Main_Page
- [11] "Judge0 - Where code happens.," *Judge0*. Accessed: Feb. 23, 2023. [Online]. Available: <https://judge0.com>
- [12] "LeetCode - The World's Leading Online Programming Learning Platform," *LeetCode*. Accessed: Feb. 23, 2023. [Online]. Available: <https://leetcode.com/>

- [13] A. Chapyzhenka and J. Probell, "Rendering Beautiful Waveforms from Plain Text," in Proceedings of the Synopsys User Group (SNUG) Silicon Valley, 2016, pp. 1-16.
- [14] "wavedrom/vcdrom: VCD viewer," *WaveDrom*. Accessed: June. 12, 2023. [Online]. Available: <https://github.com/wavedrom/vcdrom>
- [15] E. F. Heng, "Web development for auto-assessment platform," 2022, Accessed: Feb. 23, 2023. [Online]. Available: <https://dr.ntu.edu.sg/handle/10356/157189>