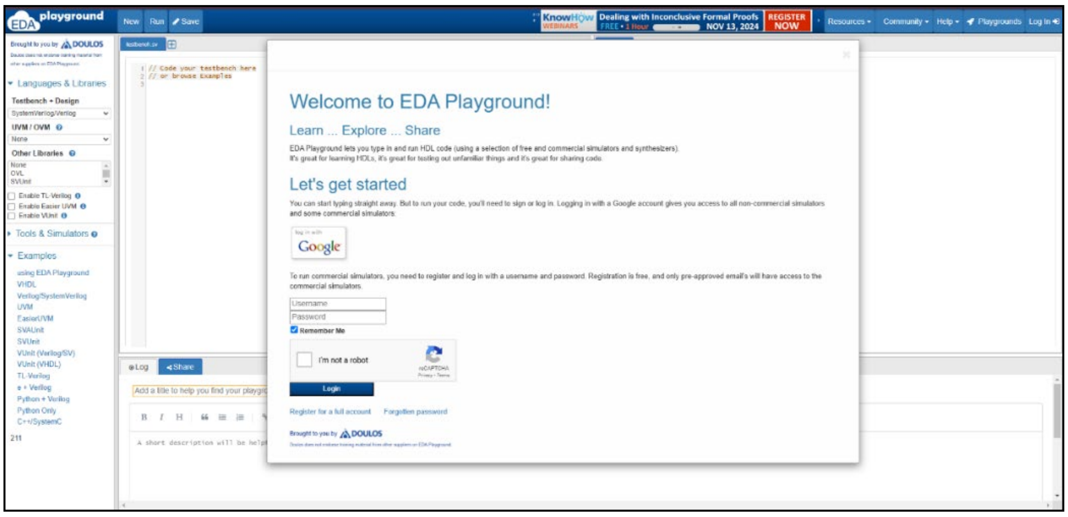
EXPERIMENT- 9
Verification Bridge Course EDA Guidelines & Lecture Highlights

1) Link to EDA Playground



Welcome to EDA Playground!

Learn ... Explore ... Share

EDA Playground lets you type in and run HDL code (using a selection of free and commercial simulators and synthesizers).
It's great for learning HDLs, it's great for testing out unfamiliar things and it's great for sharing code.

Let's get started

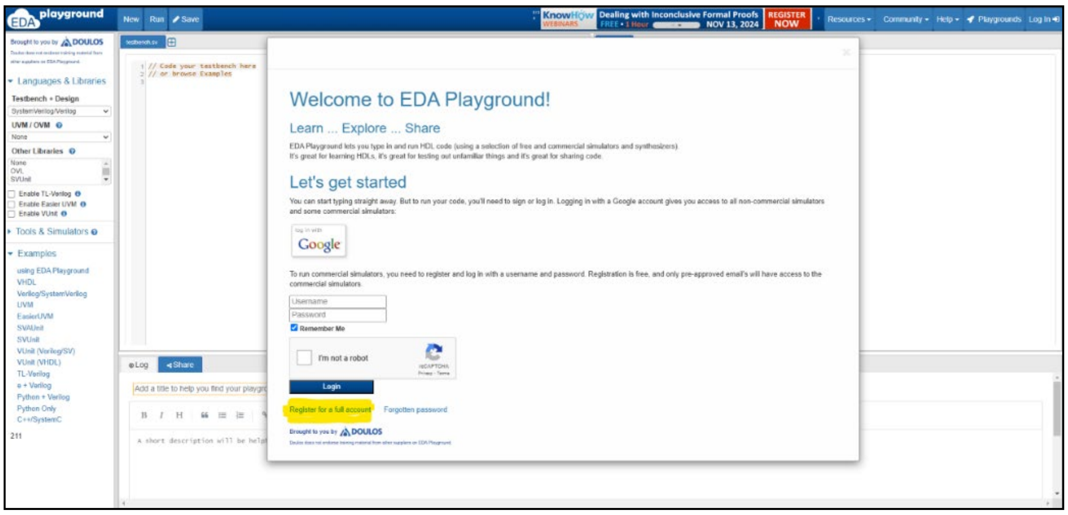
You can start typing straight away. But to run your code, you'll need to sign or log in. Logging in with a Google account gives you access to all non-commercial simulators and some commercial simulators.

To run commercial simulators, you need to register and log in with a username and password. Registration is free, and only pre-approved email's will have access to the commercial simulators.

Figure: Link to EDA Playground

2) Sign Up or Registration

After visiting the link specified in Step 1. Click on the highlighted text on the webpage. You will be directed to the EDA Playground Registration.


Welcome to EDA Playground!

Learn ... Explore ... Share

EDA Playground lets you type in and run HDL code (using a selection of free and commercial simulators and synthesizers).
It's great for learning HDLs, it's great for testing out unfamiliar things and it's great for sharing code.

Let's get started

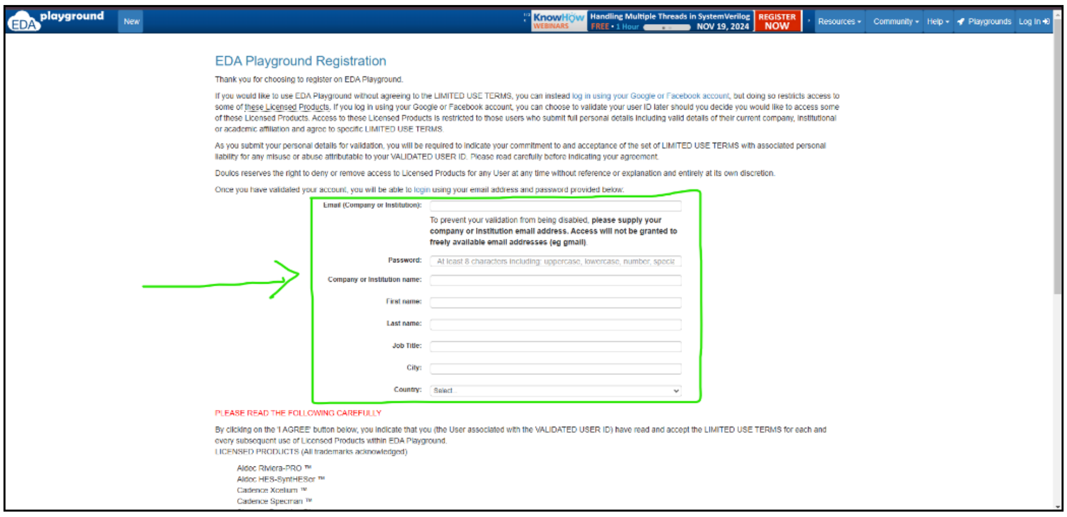
You can start typing straight away. But to run your code, you'll need to sign or log in. Logging in with a Google account gives you access to all non-commercial simulators and some commercial simulators.

To run commercial simulators, you need to register and log in with a username and password. Registration is free, and only pre-approved email's will have access to the commercial simulators.

Figure: Sign Up or Registration

3) EDA Playground Registration

Enter all the details highlighted in the Green Box in the below image. Please use your Manipal Learners ID credentials to avail full registration to EDA Playground.



EDA Playground Registration

Thank you for choosing to register on EDA Playground.

If you would like to use EDA Playground without agreeing to the LIMITED USE TERMS, you can instead log in using your Google or Facebook account, but doing so restricts access to some of these Licensed Products. If you log in using your Google or Facebook account, you can choose to validate your user ID later should you decide you would like to access some of these Licensed Products. Access to these Licensed Products is restricted to those users who submit full personal details including valid details of their current company, institutional or academic affiliation and agree to specific LIMITED USE TERMS.

As you submit your personal details for validation, you will be required to indicate your commitment to and acceptance of the set of LIMITED USE TERMS with associated personal liability for any misuse or abuse attributable to your VALIDATED USER ID. Please read carefully before indicating your agreement.

Doulos reserves the right to deny or remove access to Licensed Products for any User at any time without reference or explanation and entirely at its own discretion.

Once you have validated your account, you will be able to login using your email address and password provided below.

Email (Company or Institution):	
	To prevent your validation from being disabled, **please supply your company or institution email address. Access will not be granted to freely available email addresses (eg gmail)**.
Password:	At least 8 characters including: uppercase, lowercase, number, specia
Company or Institution name:	
First name:	
Last name:	
Job Title:	
City:	
Country:	Select..

PLEASE READ THE FOLLOWING CAREFULLY

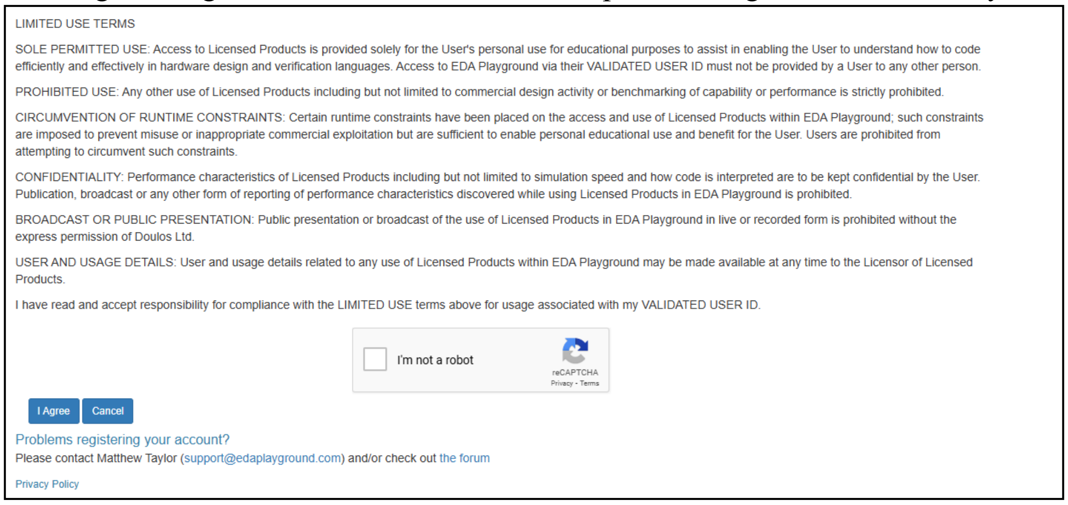
By clicking on the 'I AGREE' button below, you indicate that you (the User associated with the VALIDATED USER ID) have read and accept the LIMITED USE TERMS for each and every subsequent use of Licensed Products within EDA Playground.
LICENSED PRODUCTS (All trademarks acknowledged)

- Aldec Riviera-PRO ™
- Aldec HES-SynthHESer ™
- Cadence Xcelium ™
- Cadence Spectran ™

Figure:EDA Playground Registration

4) Accept Limited Use Terms

Please go through the Limited Use Terms and, please do agree to the same if you are willing to.



LIMITED USE TERMS

SOLE PERMITTED USE: Access to Licensed Products is provided solely for the User's personal use for educational purposes to assist in enabling the User to understand how to code efficiently and effectively in hardware design and verification languages. Access to EDA Playground via their VALIDATED USER ID must not be provided by a User to any other person.

PROHIBITED USE: Any other use of Licensed Products including but not limited to commercial design activity or benchmarking of capability or performance is strictly prohibited.

CIRCUMVENTION OF RUNTIME CONSTRAINTS: Certain runtime constraints have been placed on the access and use of Licensed Products within EDA Playground; such constraints are imposed to prevent misuse or inappropriate commercial exploitation but are sufficient to enable personal educational use and benefit for the User. Users are prohibited from attempting to circumvent such constraints.

CONFIDENTIALITY: Performance characteristics of Licensed Products including but not limited to simulation speed and how code is interpreted are to be kept confidential by the User. Publication, broadcast or any other form of reporting of performance characteristics discovered while using Licensed Products in EDA Playground is prohibited.

BROADCAST OR PUBLIC PRESENTATION: Public presentation or broadcast of the use of Licensed Products in EDA Playground in live or recorded form is prohibited without the express permission of Doulos Ltd.

USER AND USAGE DETAILS: User and usage details related to any use of Licensed Products within EDA Playground may be made available at any time to the Licensor of Licensed Products.

I have read and accept responsibility for compliance with the LIMITED USE terms above for usage associated with my VALIDATED USER ID.

Problems registering your account?
Please contact Matthew Taylor (support@edaplayground.com) and/or check out the forum

Privacy Policy

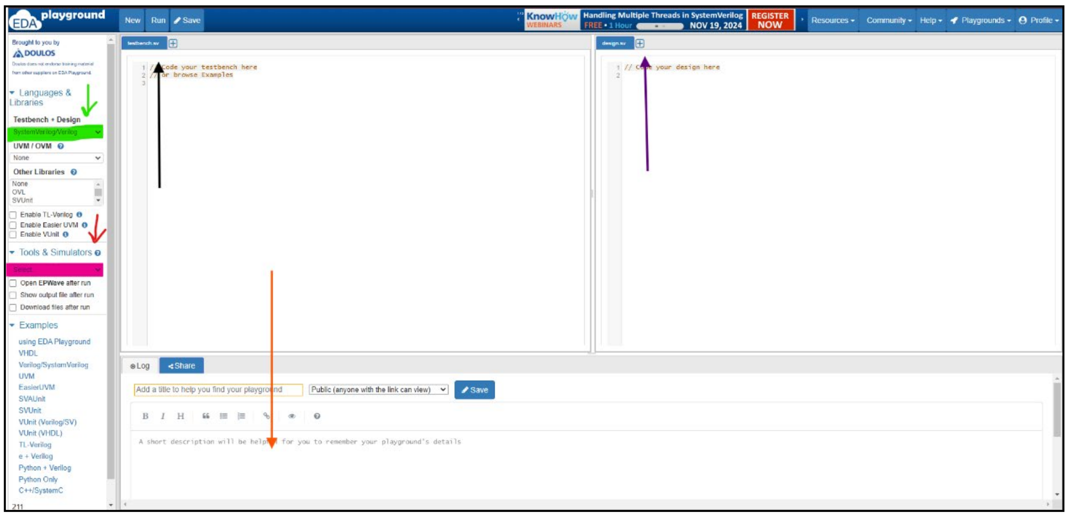
Figure:Accept Limited Terms of Use

5) Verify you EMAIL

If the registration is successful, you can expect an EMAIL Verification prior to signing in.

6) EDA Playground Startup Page

- Once you have registered; the startup page of the EDA Playground has two files. Please select the type of testbench and design you would like to code in. (as highlighted in the Languages and Libraries section).
- Please select the tools and simulators on which you would like to run the design. (as highlighted in the Tools and Simulators section).
- Please write the test bench in the section pointed in the Testbench module.
- Please write the design files in the section pointed in the Design module.
- The output and the logs can be checked in the section pointed the bottom module.



```
// Code your testbench here
// or browse Examples
```

```
// Code your design here
```

Figure 5: EDA Playground Startup

Lecture Highlights

❖ Lecture 1:

Step 1. Account Creation and Execution

- Instructions likely relate to setting up user credentials and environment necessary for accessing EDA tools.
- This section outlines the execution steps in the tool, although specifics aren’t listed.

Step 2. Signal Generation

- Discusses the generation of global signals, including clocks and resets, which are fundamental for synchronizing digital circuits.
- Distinguishes between fixed and variable rate clocks for creating different timing requirements in simulations.

Step 3. Signal Classifications

- Global Signals: Such as the main clock (clk) and reset (reset).
- Data Signals: For example, rdata (read data) and wdata (write data), used for transferring data.
- Control Signals: Control flow in a system, including signals like wr (write enable), oe (output enable), raddr (read address), and waddr (write address).

Step 4: Initial Block Usage in Testbench

- The testbench (TB) setup starts with defining an initial block using Verilog syntax.
- The timescale directive sets timing precision, typically in 1ns/1ps.

- Example code initializes registers, runs a sequence (a=1; after a delay #10; a=0;), and outlines the usage of the initial block for test signal setup.
- System tasks like $dumpfile and $dumpvars are used for waveform generation, and $monitor is used for real-time signal monitoring.
- The #200; $finish; command signals the end of execution.

Step 5: Always Block in Testbench

- Provides a template for combinational and sequential blocks using always @(argument) syntax.
- A combinational block example would be empty (driven by any change in inputs), while a sequential block typically depends on a clock signal.

Step 6: Clock Generation

- Includes generating clocks at different frequencies (e.g., 100 MHz, 50 MHz, 25 MHz) by setting the edge alignment. The document suggests aligning edges to generate specific frequencies.
- Calculation for clock period (e.g., 1000/100 = 10) is mentioned, where 10 ns is derived from 100 MHz.

Step 7: Custom Signal Generation

- Two methods are suggested:
    1. Task-based generation: Where tasks create repeatable clock sequences or signals.
    2. User-defined parameters: Allow users to customize signal properties.

❖ <u>**Lecture 2:**</u>

Step 1: Data Types

- 2-State and 4-State Data Types:
    - 2-State: Holds binary values, either 0 or 1, which are used where only definitive binary outcomes are needed.
    - 4-State: Can represent 0, 1, x (unknown), and z (high impedance), used in cases where signal states need additional information, such as undefined or disconnected states.
- Array Types:
    - Fixed Arrays: Arrays with a predefined size.
    - Queues: Dynamic arrays that allow insertion and deletion operations.
    - Dynamic Arrays: Arrays whose size can change during runtime.

- Integer Types:
    - Unsigned: Represents values from 0 to 2^n - 1 (e.g., bit[31:0] for a 32-bit unsigned integer).
    - Signed: Range from -2^(n-1) to 2^(n-1) - 1, allowing negative values.
- Specific Width Data Types:
    - 8-bit (byte), 16-bit (shortint), 32-bit (int), and 64-bit (longint), used for defining precise bit widths for variables.
- Real and Floating Types:
    - Real: Used for fixed-point numerical representation.
    - Realtime: Floating-point values that can be utilized for timing analysis in simulations.

Step 2: Assignments in HDL

- Procedural Assignment:
    - Defined within procedural blocks (e.g., always blocks) and used for variables that need to be updated sequentially.
- Continuous Assignment:
    - Assigns values continuously, generally used for combinational logic with assign statements.
- Wire and Reg:
    - Wire: Used for connecting different modules or blocks in the design, supporting only continuous assignment.
    - Reg: Holds a value across procedural assignments, used mainly in sequential logic.

Step 3: Procedural Blocks Example

- Example Code Block:
    - Demonstrates a clock-triggered counter increment:

```
always @(posedge clk)
begin
  count <= count + 1;
end
assign y = count;
```

    - This block increases count at each positive clock edge and assigns its value to y. It is essential to ensure y has the necessary width to hold the count value.

Step 4: Simulation and Variable Types

- Real-Time and System-Time Variables:
    - Used to track time in fixed or floating formats, particularly useful for time-sensitive simulations.
- Simulation-Specific Variables:
    - These include variables that adapt based on the simulation environment, which are essential for modeling complex hardware behaviors.

❖ **Lecture 3:**

Step 1: Purpose of Arrays

- Store multiple transaction data instances.
- Compare generator output vs "golden" data for Design Under Test (DUT) verification.

Step 2: Array Declaration and Initialization

- Bit Array: Example bit arr[8] can hold 8 elements, each of 1-bit size, indicating a fixed-size array.
- Automatic Sizing: bit arr[]={1,0,1,0} initializes an array and automatically sets its size based on the initial values (in this case, size 4).
- Repetitive Values: An array can be initialized with repetitive values using syntax like arr[] = '{6{1}}, which creates an array of size 6 with each element set to 1.
- Unique Values: arr = '{1,2,3,4} initializes the array with specific unique values.
- Default Initialization: arr[] = '{default:0} initializes all elements to a default value, such as 0.

Step 3: Array Functions and Access

- Size Retrieval: $size(arr) returns the size of the array.
- Accessing Elements:arr[index_value] to get elements.
- Printing Elements: %p displays all elements at once.

Step 4: Repetitive Operations on Arrays

- foreach Loop: The foreach(arr[j]) loop iterates through each element of the array, automatically incrementing j to match the array's indices.
- repeat Loop: Allows repeating a set of operations a fixed number of times, typically used when you need to replicate a specific task for each element.
- for Loop: Provides a traditional indexing approach to iterating over arrays, offering greater control over the start, end, and increment steps.