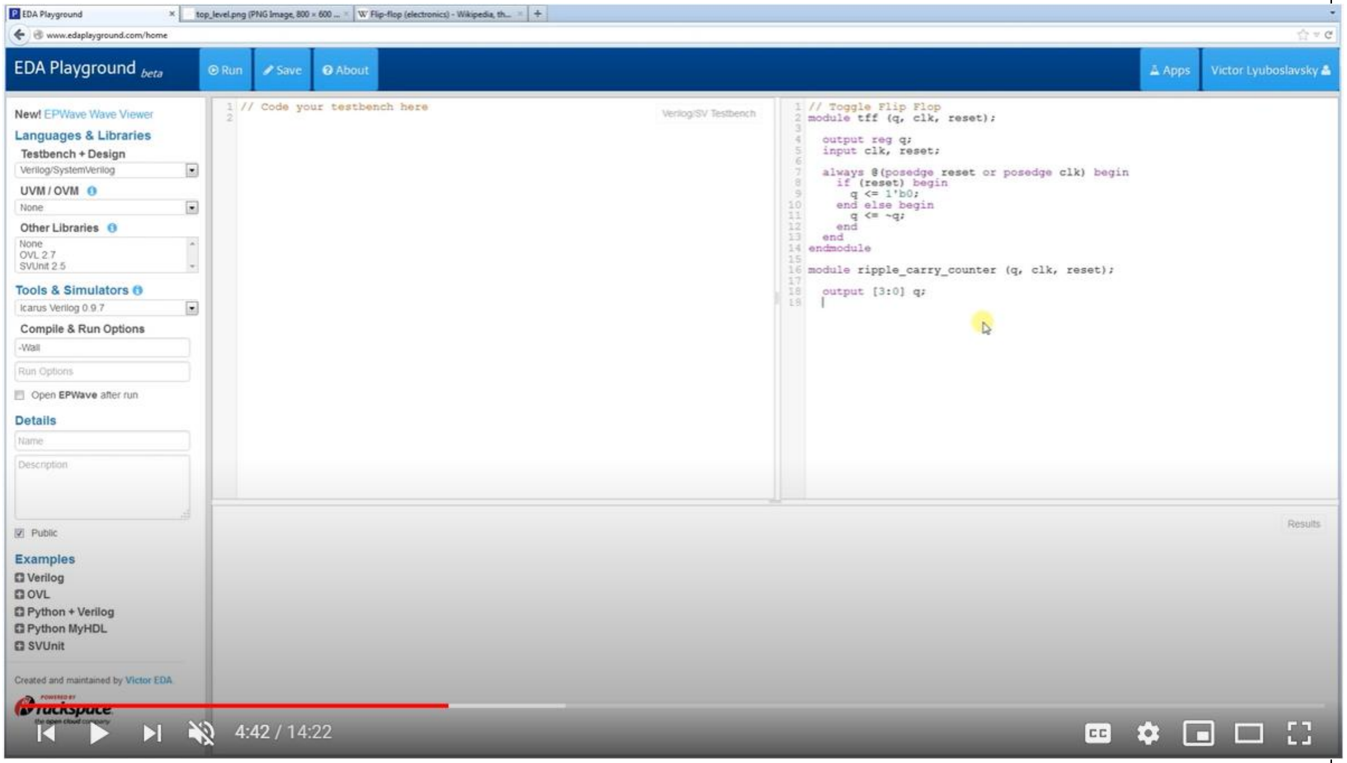
DAILY ASSESSMENT FORMAT

Date:	**04-June-2020**	**Name:**	**Shilpa S**
Course:	**HDL**	**USN:**	**4AL14EC078**
Topic:	**Hardware modelling**	**Semester & Section:**	**8th sem & 'A' section**
Github Repository:	**Shilpa_online**		

FORENOON SESSION DETAILS

Image of session



```
// Code your testbench here
```

```
// Toggle Flip Flop
module tff (q, clk, reset);

  output reg q;
  input clk, reset;

  always @(posedge reset or posedge clk) begin
    if (reset) begin
      q <= 1'b0;
    end else begin
      q <= ~q;
    end
  end
endmodule

module ripple_carry_counter (q, clk, reset);

  output [3:0] q;
  
```

Report –

In my first session today I have studied about the Hardware modelling using verilog.

ABSTRACT HARDWARE MODELLING USING AN OBJECTORIENTED LANGUAGE EXTENSION TOVHDL

Guido Schumacher, Wolfgang Nebel Department of Computer Science, Carl von Ossietzky University Oldenburg, 26111 Oldenburg, Germany ABSTRACT Reusability of hardware models becomes more and more important in the design of complex systems. This is explained for different modelling problems. Methods for re-use, which successfully use abstraction as a key to manage the

design complexity in the software domain, are analysed with respect to their applicability to hardware design. Object-oriented modelling is stated as a technique which potentially increases design productivity. Attempts to adapt such techniques for hardware specification and design are presented The lack of support for object-oriented modelling in the current version of VHDL is explained. Possible subjects for object orientation are discussed We illustrate how an object-oriented extension to the hardware description language VHDL could be used to describe hardware systems at a high level of abstraction. An explanation is given that it is not sufficient to introduce object-oriented language constructs into a parallel hardware description language, it is also necessary "to provide high-level communication constructs for the data exchange between the parallel processes.

INTRODUCTION

The non-saturating integration density of integrated circuits (despite of earlier assumptions) provides sufficient motivation to investigate further possibilities to increase design productivity. The increase of complexity can be estimated at about ten every seven years. It cannot be compensated by the increase of computing power, which on the one hand does not achieve an increase in intellectual value addition without a change in design methodology, while on the other is consumed mostly by complex data management systems and user-friendly graphical interfaces.

As keys to manage the design complexity and the increase of design effort typically the terms hierarchy and abstraction are mentioned. A hierarchical, structural decomposition of the design problem reduces the complexity by isolating subproblems and hence makes the global design problem accessible for a solution. This achievement has to be paid for with a suboptimal total solution due to the local optimization of the substructures. Abstraction should be looked at from different points of view. First, it allows us to encapsulate existing components which can be described by an abstract model containing only that information which is required at the higher level of abstraction. Examples of this kind of abstraction are cell libraries of ASIC vendors. Here abstraction allows the re-use of hardware components. They do not need to be redesigned for each application, i.e. the design cost (measured in terms of transistors) is reduced.

On the other hand, abstraction can be used for reducing design effort if design detail can automatically be attributed to less detailed design specifications using synthesis tools. Application examples are: logic synthesis, technology mapping, place and route. Here the reduction in design effort is due to a re-use of automated design strategies and architectures. At the system level, abstraction is required to describe a first specification without confusing and often unknown implementation details.

It allows a first check of the system description for inconsistencies and errors and therefore reduces the risk of redesign steps in further design phases. 6.1.1. Traditional Approaches of Re-Use In the past, the method of design data encapsulation was developed in an evolutionary way in the direction of higher levels of abstraction.

Prototyping LDPC Codes in Hardware

Any hardware design that is intended for practical applications requires implementation of prototype models on the hardware for testing [13]. Field Programmable Gate Array (FPGA) is a very flexible and widely used platform for rapid prototyping, and is also a low-cost approach compared to ASIC implementation.

FPGA is an integrated circuit that contains large numbers of identical logic cells that can be interconnected by a matrix of wires using programmable switch boxes [14]. A design can be implemented by specifying the simple logic function for each cell and selectively closing the switches in the interconnect matrix. The array of logic cells and the mesh of interconnecting wires form the basic building block of an FPGA. Complex designs can be implemented by programming these basic building blocks [15].

FPGAs offer a number of benefits over other implementation flows such as ASIC and off-the-shelf DSP and microcontroller chips. Some of the benefits of using FPGA are as follows :

1. Performance: FPGAs offer logic structures that provide the advantage of incorporating parallelism in designs and thereby significantly enhance the computational speed compared to processor-based platforms.
2. Reliability: Processor-based designs operate on instructions to perform a particular task using shared hardware resources. However, FPGA-based designs consist of dedicated hardware for performing such tasks with predictable delays. Hence, increasing the reliability of real-time systems.

3. Long-term maintenance: FPGAs provide flexibility in upgrading the design in case of a change in specification of an application over time. The time spent in redesigning/enhancing a FPGA-based design is much less compared to that of ASIC design.

4. Cost: The Non-Recurring Engineering (NRE) cost for designing a custom ASIC is huge compared to FPGA-based solutions. FPGAs bypass the backend physical design, fabrication, and packaging cost as well.

5. Time to Market: FPGA technology provides flexibility for rapid prototyping of the design by avoiding fabrication and other processing delays, thus facilitating quicker “time to market” solutions.

The major limitation of FPGA-based design is in its overall performance compared to ASIC solutions. However, it provides the best methodology for quick prototyping and testing of the designs including the above listed advantages.

Implement a simple T Flipflop and test the module using a compiler.

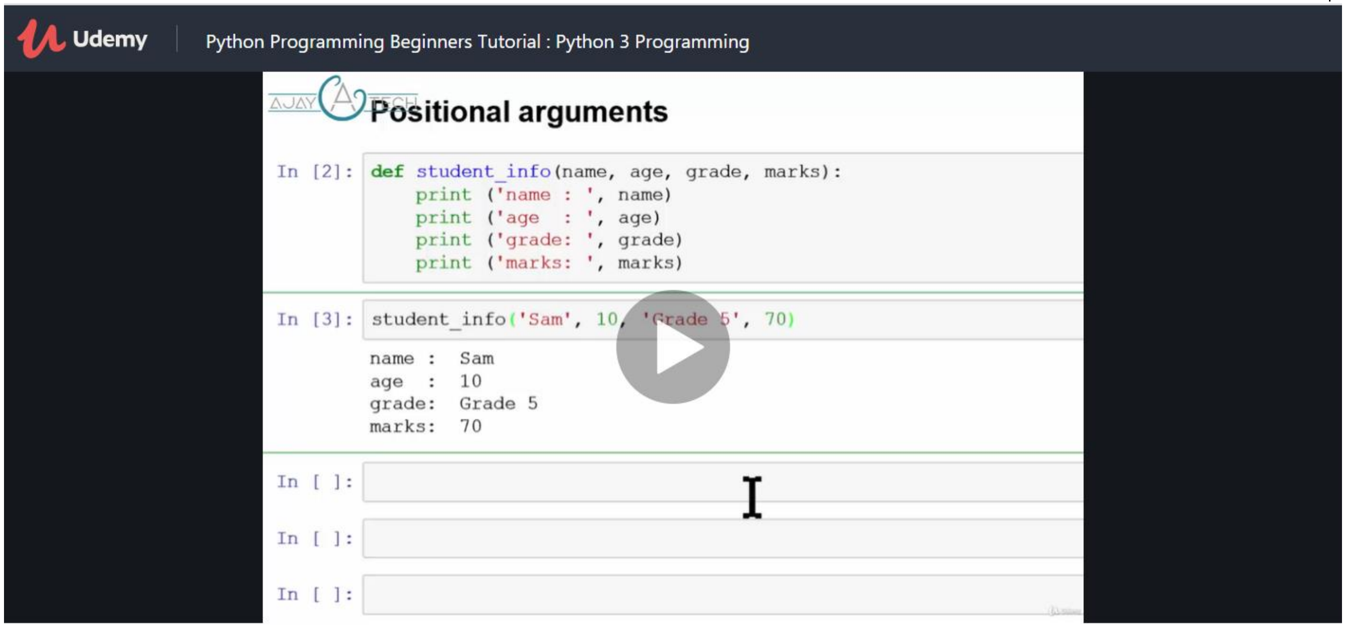
```
module tff (   input clk,
              input rstn,
              input t,
            output reg q);

  always @ (posedge clk) begin
    if (!rstn)
      q <= 0;
    else
      if (t)
          q <= ~q;
      else
          q <= q;
  end
endmodule
```

Date:	**04-June-2020**	**Name:**	**Shilpa S**
Course:	**Udemy Python Course**	**USN:**	**4AL14EC078**
Topic:	**Application 8: Build a web based financial graph**	**Semester & Section:**	**8th sem & ‘A’ section**

AFTERNOON SESSION DETAILS

Image of session



Udemy
Python Programming Beginners Tutorial : Python 3 Programming
AJAY TECH
Positional arguments

```
In [2]: def student_info(name, age, grade, marks):
            print ('name : ', name)
            print ('age  : ', age)
            print ('grade: ', grade)
            print ('marks: ', marks)

In [3]: student_info('Sam', 10, 'Grade 5', 70)

        name :  Sam
        age  :  10
        grade:  Grade 5
        marks:  70

In [ ]:

In [ ]:

In [ ]:
```

Bokeh is a powerful open source Python library that allows developers to generate JavaScript data visualizations for their web applications without writing any JavaScript. While learning a JavaScript-based data visualization library like d3.js can be useful, it's often far easier to knock out a few lines of Python code to get the job done. With Bokeh, we can create incredibly detailed interactive visualizations, or just traditional ones like the following bar chart

Bokeh provides a variety of ways to embed plots and data into HTML documents. First, a reminder of the distinction between standalone documents and apps.

Standalone Documents:

These are Bokeh documents that are not backed by a Bokeh server. They may have many tools and interactions (e.g. from CustomJS callbacks) but are self-contained HTML, JavaScript, and CSS. They can be embedded into other HTML pages as one large document, or as a set of sub-components templated individually.

Bokeh Applications:
These are Bokeh documents that are backed by a Bokeh Server. In addition to all the features of standalone documents, it is also possible to connect events and tools to real Python callbacks that execute in the Bokeh server.

HTML files Bokeh can generate complete HTML pages for Bokeh documents using the file_html() function. This function can emit HTML from its own generic template, or a template you provide. These files contain the data for the plot inline and are completely transportable, while still providing interactive tools (pan, zoom, etc.) for your plot. Here is an example:

```
from bokeh.plotting import figure
from bokeh.resources import CDN
from bokeh.embed import file_html
plot = figure()
plot.circle([1,2], [3,4])
html = file_html(plot, CDN, "my plot")
```

The returned HTML text can be saved to a file using standard python file operations. You can also provide your own template and pass in custom, or additional, template variables. See the file_html() documentation for more details. G This is a fairly low-level, explicit way to generate an HTML file, which may be useful for use from a web application, e.g. a Flask app. When using the bokeh.plotting interface in a script or Jupyter notebook, users will typically call the function output_file() in conjunction with show() or save() instead

Plots and data in the form of standalone documents as well as Bokeh applications can be embedded in HTML documents. Standalone document is a Bokeh plot or document not backed by Bokeh server. The interactions in such a plot is purely in the form of custom JS and not Pure Python callbacks. Bokeh plots and documents backed by Bokeh server can also be embedded. Such documents contain Python callbacks that run on the server. In case of standalone documents, a raw HTML code representing a Bokeh plot is obtained by file_html() function.

```
from bokeh.plotting import figure
from bokeh.resources import CDN
from bokeh.embed import file_html
fig = figure()
fig.line([1,2,3,4,5], [3,4,5,2,3])
string = file_html(plot, CDN, "my plot")
```

Return value of file_html() function may be saved as HTML file or may be used to render through URL routes in Flask app. In case of standalone document, its JSON representation can be obtained by json_item() function.

```
from bokeh.plotting import figure
from bokeh.embed
import file_html
import json
fig = figure()
fig.line([1,2,3,4,5], [3,4,5,2,3])
item_text = json.dumps(json_item(fig, "myplot"))
```

This output can be used by the Bokeh.embed.embed_item function on a webpage –

```
item = JSON.parse(item_text);
Bokeh.embed.embed_item(item);
```

Bokeh applications on Bokeh Server may also be embedded so that a new session and Document is created on every page load so that a specific, existing session is loaded. This can be accomplished with the server_document() function. It accepts the URL to a Bokeh server application, and returns a script that will embed new sessions from that server any time the script is executed