DAILY ASSESSMENT FORMAT

Date:	**05/06/2020**	**Name:**	**Lavanya B**
Course:	**Verilog HDL**	**USN:**	**4al17ec043**
Topic:	**Verilog Building project using FPGA**	**Semester & Section:**	**6th A**
Github Repository:	**Lavanya-B**		

FORENOON SESSION DETAILS
Image of session

Fall Delay

The fall delay is associated with a gate output transition to 0 from a

Turn–off Delay

The Turn–off delay is associated with a gate output transition to z fi

Min Value

The min value is the minimum delay value that the gate is expectec

Typ Value

The typ value is the typical delay value that the gate is expected to

Max Value

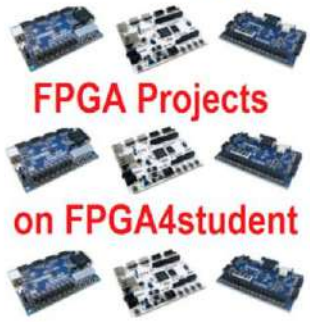
The max value is the maximum delay value that the gate is expecte

Examples

www.asic–world.com GATE LEVEL MODELING

FPGA Projects

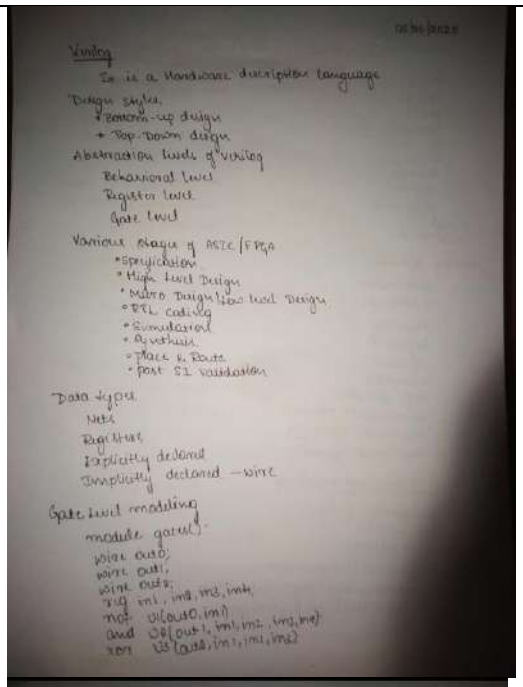
This page presents FPGA projects on fpga4student.com. The first FPGA project helps students understand the basics of FPGAs and how Verilog/ VHDL works on FPGA.



Some of the FPGA projects can be FPGA tutorials such as What is FPGA Programming, image processing on FPGA, matrix multiplication on FPGA Xilinx using Core Generator, Verilog vs VHDL: Explain by Examples and how to load text files or images into FPGA. Many others FPGA projects provide students with full Verilog/ VHDL source code to practice and run on FPGA boards. Some of them can be used for another bigger FPGA projects.

Following are the FPGA projects on

Report



Verilog

Is it a Hardware discription language

Design styles.
- Bottom-up design
- Top-Down design

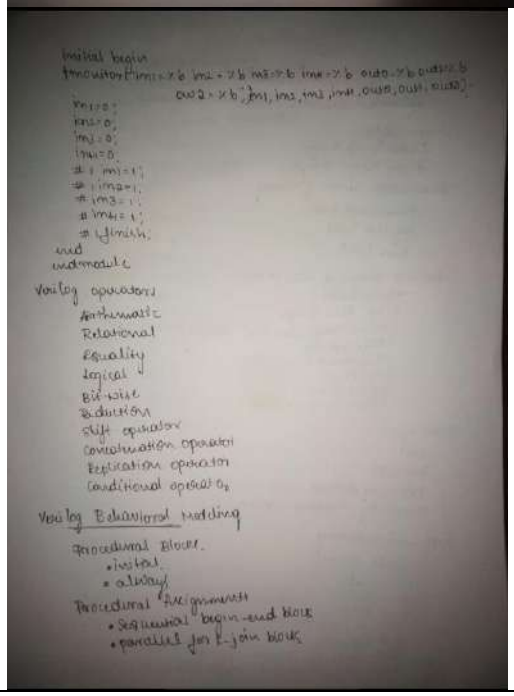
Abstraction levels of Verilog
- Behavioral level
- Register level
- Gate level

Various stages of ASIC/FPGA
- Specification
- High level Design
- Micro Design/Low level Design
- RTL coding
- Simulation
- Synthesis
- Place & Route
- Post SI validation

Data types
- Nets
- Registers
  - Explicitly declared
  - Implicitly declared — wire

Gate level modeling

```
module gate();
wire out0;
wire out1;
wire out2;
reg in1, in2, in3, in4;
not U1(out0, in1);
and U0(out1, in1, in2, in3, in4);
xor U3(out2, in1, in2, in3);
```



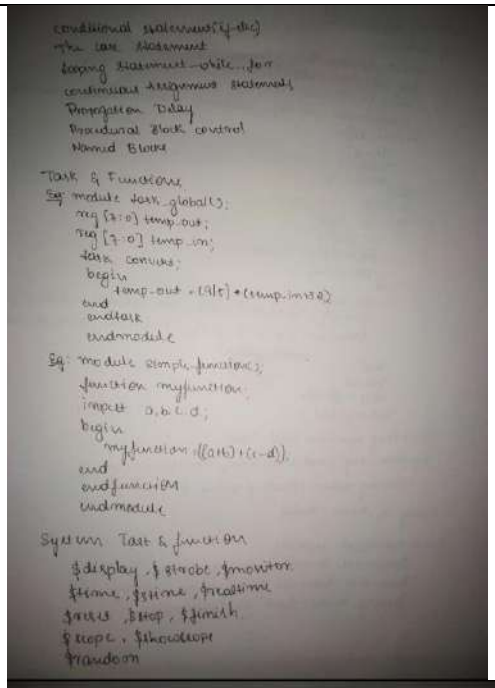
```
initial begin
$monitor($time,"in1=%b in2=%b in3=%b in4=%b out0=%b out1=%b
          out2=%b",in1, in2, in3, in4, out0, out1, out2);
in1=0;
in2=0;
in3=0;
in4=0;
#1 in1=1;
#1 in2=1;
#1 in3=1;
#1 in4=1;
#1 $finish;
end
endmodule
```

Verilog operators
- Arithmetic
- Relational
- Equality
- Logical
- Bit-wise
- Reduction
- Shift operators
- Concatenation operator
- Replication operator
- Conditional operator

Verilog Behavioral modeling

Procedural Blocks
- initial
- always

Procedural Assignments
- Sequential begin-end block
- parallel fork-join block

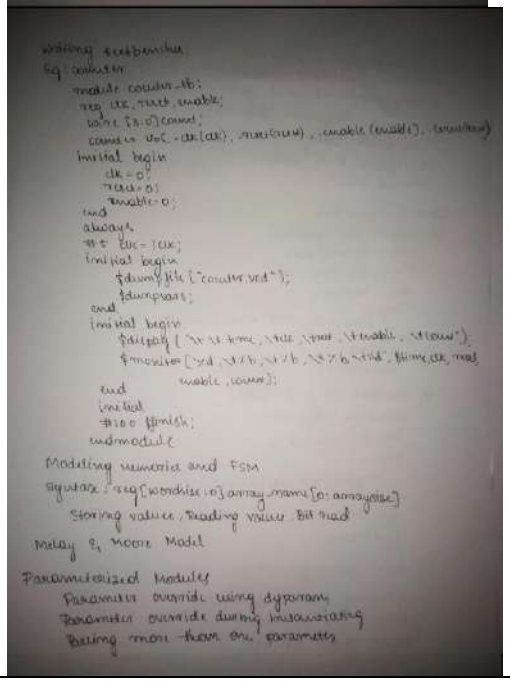


conditional statements(if-else)
The case statement
looping statement - while, for
continuous assignment statement
Propagation Delay
Procedural Block control
Named Blocks

Task & Functions
Eg: module task_globals;
reg [7:0] temp_out;
reg [7:0] temp_in;
task convert;
begin
temp_out = (9/5) * (temp_in+32)
end
endtask
endmodule

Eg: module simple_function;
function myfunction;
input a,b,c,d;
begin
myfunction = ((a+b)+(c-d));
end
endfunction
endmodule

System Task & function
$display, $strobe, $monitor
$time, $stime, $realtime
$reset, $stop, $finish
$scope, $showscope
$random

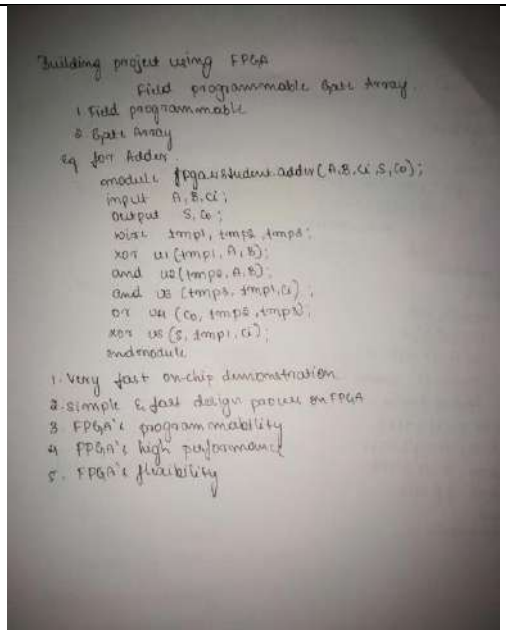


writing testbenches
Eg: counter
module counter_tb;
reg clk, reset, enable;
wire [3:0] count;
counter UUT - clk(clk), .reset(rst), .enable(enable), .count(count)
initial begin
clk = 0;
reset = 0;
enable = 0;
end
always
#5 clk = !clk;
initial begin
$dumpfile("counter.vcd");
$dumpvars;
end
initial begin
$display("\n\t time \t clk \t rst \t enable \t count");
$monitor("%d %b %b %b %d", $time, clk, reset,
enable, count);
end
initial
#100 $finish;
endmodule

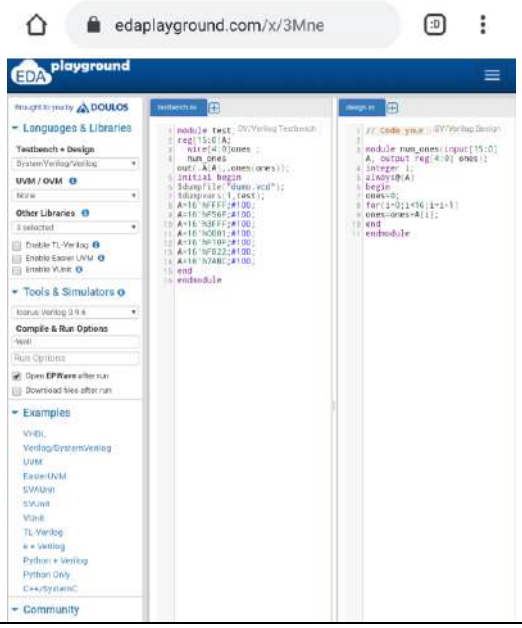
Modeling memories and FSM
Syntax: reg[wordsize:0] array_name[0:arraysize]
Storing values, Reading values, Bit read

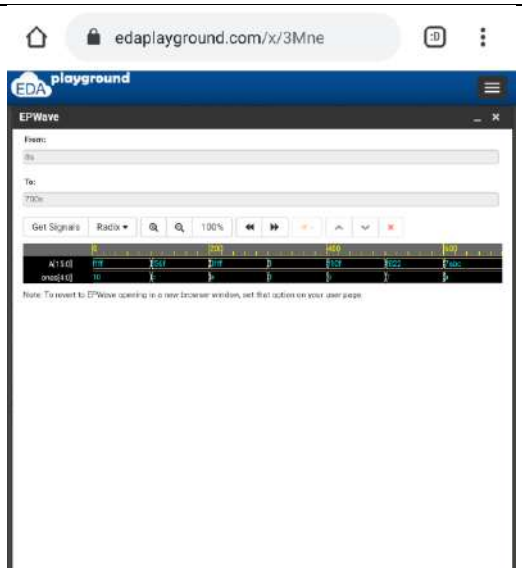
Mealy & Moore Model

Parameterized modules
Parameter override using dparam
Parameter override during instantiating
Having more than one parameters


Building project using FPGA
Field programmable Gate Array
1. Field programmable
2. Gate Array
eg for Adder
module fpga_u1Student.addw(A,B,Ci,S,Co);
input A,B,Ci;
output S,Co;
wire tmp1, tmp2, tmp3;
xor u1(tmp1, A, B);
and u2(tmp2, A, B);
and u3(tmp3, tmp1, Ci);
or u4(Co, tmp2, tmp3);
xor u5(S, tmp1, Ci);
endmodule
1. Very fast on-chip demonstration
2. Simple & fast design process on FPGA
3. FPGA's programmability
4. FPGA's high performance
5. FPGA's flexibility

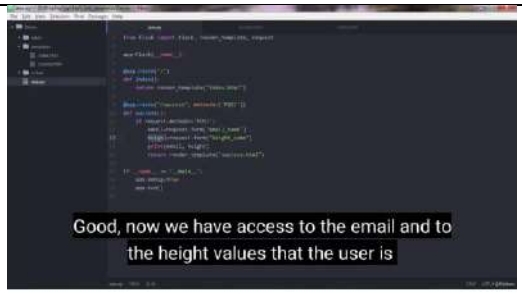
Task


edaplayground.com/x/3Mne
EDA playground
Languages & Libraries
Tools & Simulators
Examples
Community



Date:	**05/06/2020**	**Name:**	**Lavanya B**
Course:	**Python**	**USN:**	**4al17ec043**
Topic:	**Application 10**	**Semester & Section:**	**6th A**

AFTERNOON SESSION DETAILS
Image of session



Lectures More

Video - 08:31 mins - Resources (1)

Section 32 - Application 10: Build a Data Collector Web App with PostGre...

258 Data Collector Web App - How The Output Wi...
Video - 02:59 mins - Resources (1)

259 PostGreSQL Database Web App with Flask: S...
Video - 06:08 mins

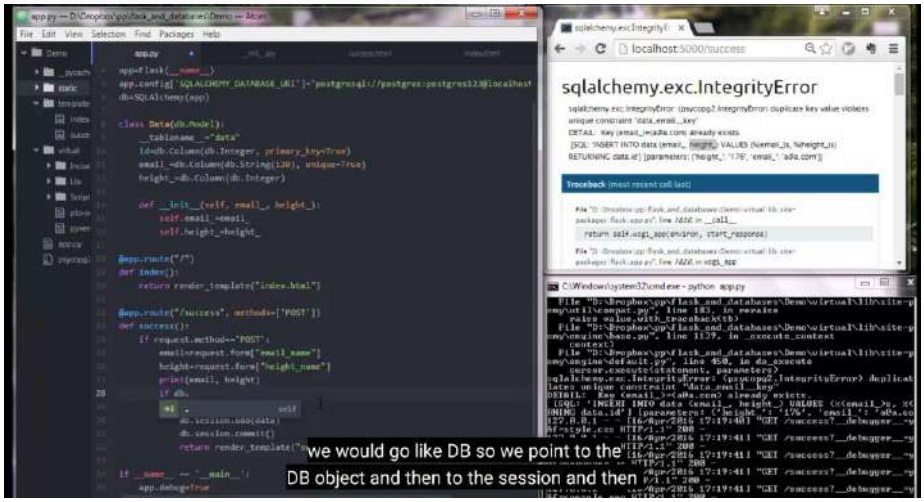
260 Frontend: HTML Part
Video - 14:52 mins

261 Frontend: CSS Part
Video - 10:11 mins

262 Backend: Getting User Input
Video - 17:31 mins

263 Backend: The PostGreSQL Database Model
Video - 18:17 mins

264 Backend: Storing User Data to the Database
Video - 19:14 mins



Report

Application 10: Build a data collector web application with post GRESQL and flask

Flask start up and configuration

Most widely used Python libraries, the Flask package is installable from the Python Package
Index (PPI). First create a directory to work in (something like flask_ to do is a fine directory name)
then install the flask package. You'll also want to install flask- so your Flask application has a simple way to talk to a SQL data base. A good way to get moving is to turn the codebase into an installable Python distribution.

At the project's root, create setup.py and a directory called to do to hold the
source code.
The setup.py should look something like this:
requires =['flask', 'flask-sqlal chemy', 'psycopg2',]

setup(name='flask_todo', version='0.0', description='A)

To-Do List built with Flask',
author='<Your actual name here>', author_email='<Your actual e-mail address here>',
keywords='web flask',
packages=find_packages(),
include_package_data=True,
install_requires=requires

This way, whenever you want to install or deploy your project, you'll have all the necessary packages in the requires list. You'll also have everything you need to set up and install the package in site packages.
For more information on how to write an installable Python distribution, check out the
docs on setup.py.
Within the to do directory containing your source code, create an app.py file and a
blank __init__.py file.
The __init__.py file allows you to import from to do as if it were an installed package. The app.py file will be the application's root. This is where all the Flask application goodness will go, and you'll create an environment variable that points to that file. If you're using pipenv (like I am), you can locate your virtual environment with pipenv --venv and set up that environment variable in your environment's activate script.