Volume

1

FPGA-EMBEDDED LAB

College of Engineering Pune

Lab Manual

VIRTUAL LABS PROJECT

FPGA-Embedded Lab Manual

Lab In-Charge

D.N Sonawane
dns.instru@coep.ac.in

Kalyani Bhole
kab.instru@coep.ac.in

Table of Contents

1.1 Evolution of Programmable Logical devices

This chapter deals with the history and advent of era of programmable logical devices.

The first type of user-programmable chip that could implement logic circuits was the Programmable Read-Only Memory (PROM), in which address lines can be used as logic circuit inputs and data lines as outputs. Logic functions, however, rarely require more than a few product terms, and a PROM contains a full decoder for its address inputs. PROMS are thus an inefficient architecture for realizing logic circuits, and so are rarely used in practice for that purpose. The first device developed later specifically for implementing logic circuits was the Field-Programmable Logic Array (FPLA), or simply PLA for short. A PLA consists of two levels of logic gates: a programmable "wired" AND-plane followed by a programmable "wired" OR-plane. A PLA is structured so that any of its inputs (or their complements) can be AND'ed together in the AND-plane; each AND-plane output can thus correspond to any product term of the inputs. Similarly, each OR plane output can be configured to produce the logical sum of any of the AND-plane outputs. With this structure, PLAs are well-suited for implementing logic functions in sum-of-products form. They are also quite versatile, since both the AND terms and OR terms can have many inputs (this feature is often referred to as wide AND and OR gates).
When PLAs were introduced in the early 1970s, by Philips, their main drawbacks were that they were expensive to manufacture and offered somewhat poor speed-performance. Both disadvantages

were due to the two levels of configurable logic, because programmable logic planes were difficult to manufacture and introduced significant propagation delays. To overcome these weaknesses, Programmable Array Logic (PAL) devices were developed. To compensate for lack of generality incurred because the OR- plane is fixed, several variants of PALs are produced, with different numbers of inputs and outputs, and various sizes of OR-gates. PALs usually contain flip-flops connected to the OR-gate outputs so that sequential circuits can be realized. PAL devices are important because when introduced they had a profound effect on digital hardware design, and also they are the basis for some of the newer, more sophisticated architectures that will be described shortly. Variants of the basic PAL architecture are featured in several other products known by different acronyms. All small PLDs, including PLAs, PALs, and PAL-like devices are grouped into a single category called Simple PLDs (SPLDs), whose most important characteristics are low cost and very high pin-to-pin speed-performance. As technology has advanced, it has become possible to produce devices with higher capacity than SPLDs. The difficulty with increasing capacity of a strict SPLD architecture is that the structure of the programmable logic-planes grows too quickly in size as the number of inputs is increased. The only feasible way to provide large capacity devices based on SPLD architectures is then to integrate multiple SPLDs onto a single chip and provide interconnect to programmably connect the SPLD blocks together. Many commercial FPD products exist on the market today with this basic structure, and are collectively referred to as Complex PLDs (CPLDs). CPLDs were pioneered by Altera, first in their family of chips called Classic EPLDs, and then in three additional series, called MAX 5000, MAX 7000 and MAX 9000. Because of a rapidly growing market for large FPDs, other manufacturers developed devices in the CPLD category and there are now many choices available. All of the most important commercial products will be described in Section 2. CPLDs provide logic capacity up to the equivalent of about 50 typical SPLD devices, but it is somewhat difficult to extend these architectures to higher densities. To build FPDs with very high logic capacity, a different approach is needed. The highest capacity general purpose logic chips available today are the traditional gate arrays sometimes

referred to as Mask-Programmable Gate Arrays (MPGAs). MPGAs consist of an array of pre-fabricated transistors that can be customized into the user's logic circuit by connecting the transistors with custom wires. Customization is performed during chip fabrication by specifying the metal interconnect, and this means that in order for a user to employ an MPGA a large setup cost is involved and manufacturing time is long. Although MPGAs are clearly not FPDs, they are mentioned here because they motivated the design of the user-programmable equivalent: Field-Programmable Gate Arrays (FPGAs). Like MPGAs, FPGAs comprise an array of uncommitted circuit elements, called logic blocks, and interconnect resources, but FPGA configuration is performed through programming by the end user. An illustration of a typical FPGA architecture appears in Figure 2. As the only type of FPD that supports very high logic capacity, FPGAs have been responsible for a major shift in the way digital circuits are designed.

1.2 Introduction to FPGA

What is an FPGA Exactly?

An **FPGA** or a Field-Programmable Gate Array is a Field Programmable Device featuring a general structure that allows very high logic capacity. Whereas CPLDs feature logic resources with a wide number of inputs (AND planes), FPGAs offer more narrow logic resources. FPGAs also offer a higher ratio of flip-flops to logic resources than do CPLDs. FPGAs comprise an array of uncommitted circuit elements, called *logic blocks,* and interconnect resources, but FPGA configuration is performed through programming by the end user. As the only type of FPD that supports very high logic capacity, FPGAs have been responsible for a major shift in the way digital circuits are designed.

1.3 Commercially available FPGAs

There are two basic categories of FPGAs on the market today: 1. SRAM-based FPGAs and 2. antifuse-based FPGAs. In the first category, Xilinx and Altera are the leading manufacturers in terms of number of users, with the major competitor being AT&T. For antifuse-based products, Actel, Quicklogic and Cypress, and Xilinx offer competing products.

1.4 Xilinx SRAM-based FPGAs

The basic structure of Xilinx FPGAs is array-based, meaning that each chip comprises a two dimensional array of logic blocks that can be interconnected via horizontal and vertical routing channels. Xilinx introduced the first FPGA family, called the XC2000 series, in about 1985 and now offers more than six generations. We will focus on the most widely used and more popular Spartan 3E family. The Spartan-3E family of Field-Programmable Gate Arrays (FPGAs) is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. The five-member family offers densities ranging from 100,000 to 1.6 million system gates.

Spartan-3E family architecture consists of five fundamental programmable functional elements:

1. **Configurable Logic Blocks (CLBs)** contain flexible Look-Up Tables (LUTs) that implement logic plus storage elements used as flip-flops or latches. CLBs perform a wide variety of logical functions as well as store data.

2. **Input/Output Blocks (IOBs)** control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation. Supports a variety of signal standards, including four high- performance differential standards. Double Data-Rate (DDR) registers are included.

3. **Block RAM** provides data storage in the form of 18-Kbit dual-port blocks.

4. **Multiplier Blocks** accept two 18-bit binary numbers as inputs and calculate the product.
5. **Digital Clock Manager (DCM)** Blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase-shifting clock signals.

These elements are organized as shown in Figure 1. A ring of IOBs surrounds a regular array of CLBs. Each device has two columns of block RAM except for the XC3S100E, which has one column. Each RAM column consists of several 18-Kbit RAM blocks. Each block RAM is associated with a dedicated multiplier. The DCMs are positioned in the center with two at the top and two at the bottom of the device. The XC3S100E has only one DCM at the top and bottom, while the XC3S1200E and XC3S1600E add two DCMs in the middle of the left and right sides.

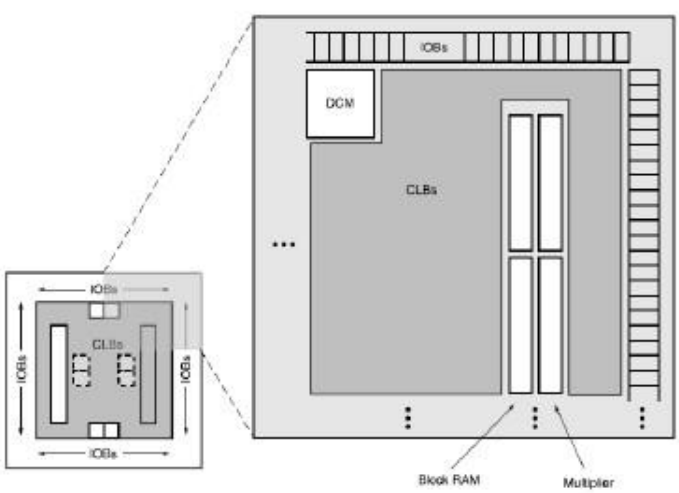


Figure 1 Internal Architecture of FPGA

1.5 Xilinx Spartan 3E Development Board

The Xilinx Spartan-3E Starter Kit as shown in figure 2 highlights the unique features of the Spartan-3E FPGA family and provides a convenient development board for embedded processing applications.

Figure 2 Xilinx Spartan 3E development board

The key features of the Spartan-3E development board are:

- Xilinx XC3S500E Spartan-3E FPGA
  - Up to 232 user-I/O pins
  - 320-pin FBGA package
  - Over 10,000 logic cells
- Xilinx 4 Mbit Platform Flash configuration PROM
- Xilinx 64-macrocell XC2C64A CoolRunner™ CPLD
- 64 MByte (512 Mbit) of DDR SDRAM, x16 data interface, 100+ MHz

- 16 MByte (128 Mbit) of parallel NOR Flash (Intel StrataFlash)
    - FPGA configuration storage
    - MicroBlaze code storage/shadowing
- 6 Mbits of SPI serial Flash (STMicro)
    - FPGA configuration storage
    - MicroBlaze code shadowing
- 2-line, 16-character LCD screen
- PS/2 mouse or keyboard port
- VGA display port
- 10/100 Ethernet PHY (requires Ethernet MAC in FPGA)
- Two 9-pin RS-232 ports (DTE- and DCE-style)
- On-board USB-based FPGA/CPLD download/debug interface
- 50 MHz clock oscillator
- SHA-1 1-wire serial EEPROM for bitstream copy protection
- Hirose FX2 expansion connector
- Three Digilent 6-pin expansion connectors
- Four-output, SPI-based Digital-to-Analog Converter (DAC)
- Two-input, SPI-based Analog-to-Digital Converter (ADC) with programmable-gain
- pre-amplifier
- ChipScope™ SoftTouch debugging port
- Rotary-encoder with push-button shaft
- Eight discrete LEDs
- Four slide switches
- Four push-button switches
- SMA clock input
- 8-pin DIP socket for auxiliary clock oscillator

1.6 Working with Spartan 3E FPGA

1.6.1 There are numerous development languages available for programming FPGA. Some of the standard languages are as follows:

1. VHDL
2. Verilog
3. Handle-C etc

1.6.2 The typical development cycle is as follows:

1. Develop the FPGA VHDL/verilog code
2. Synthesis the code
3. Simulate your FPGA design
4. Place, map and route the design
5. Generate the bit file
6. Download the bit file to the FPGA hardware

1.6.3 Development toolsets available to program FPGA are as follows:

1. Xilinx's ISE
2. System generator
3. EDK
4. Chip scope pro etc

1.6.4 Verilog Programming

Basic syntax of a standard Verilog program is:

```
module <module name>(input,output);
        $declaration section
        $Procedural Blocks
end module
```

Any program of the verilog starts with the *module<module name>* followed by input, output section.
Input, output section includes the hardware inputs and outputs of the program code.

Declaration section : In verilog declaration can be of two types

1. Nets - represents structural connections between components.
2. Registers - represent variables used to store data.

Types of Nets

wire	Interconnecting wire - no special resolution function
wor	Wired outputs OR together
wand	Wired outputs AND together
tri0	Net pulls-down or pulls-up when not driven
supply0	Net has a constant logic 0 or logic 1

Note

Of all net types, wire is the most widely used

Register Data Types

- Registers store the last value assigned to them until another assignment statement changes their value.
- Registers represent data storage constructs.
- You can create arrays of the regs called memories.
- Register data types are used as variables in procedural blocks.
- A register data type is required if a signal is assigned a value within a procedural block
    - Procedural blocks begin with keyword *initial* and *always.*

Procedural Blocks

Verilog behavioral code is inside procedures blocks, but there is a exception, some behavioral code also exist outside procedures blocks.

There are two types of procedural blocks in Verilog

- **initial** : initial blocks execute only once at time zero (start execution at time zero).
- **always** : always blocks loop to execute over and over again, in other words as name means, it executes always.

Examples:-

initial

```
Initial
begin
clk = 0;
reset = 0;
enable = 0;
data = 0;
end
```

always

```
always @
(posedge clk)
begin : D_FF
if (reset == 1)
q <= 0;
else
q <=d;
end
```

If a procedure block contains more then one statement, those statements must be enclosed within

- Sequential **begin - end** block
- Parallel **fork - join** block

1.6.5 Hello world program using Verilog

```
//------------------------------------------------------------
// This is my first Verilog Program
// Design Name : hello_world
// File Name : hello_world.v
// Function : This program will print "hello world"
// Try this program using FPGA-Verilog simulator and get the
// output as "Hello World" in output window.
//------------------------------------------------------------

module hello_world;
initial begin
$display ("Hello World");
$finish;
end
endmodule // End of Module hello_world
```

1.7 Steps to perform Experiment using FPGA-Verilog Simulator

1.7.1 Introduction of FPGA-Verilog Simulator

FPGA-Verilog simulator is composed of following sections:

1. Input window
2. Editor window
3. Output window
4. Simulation window

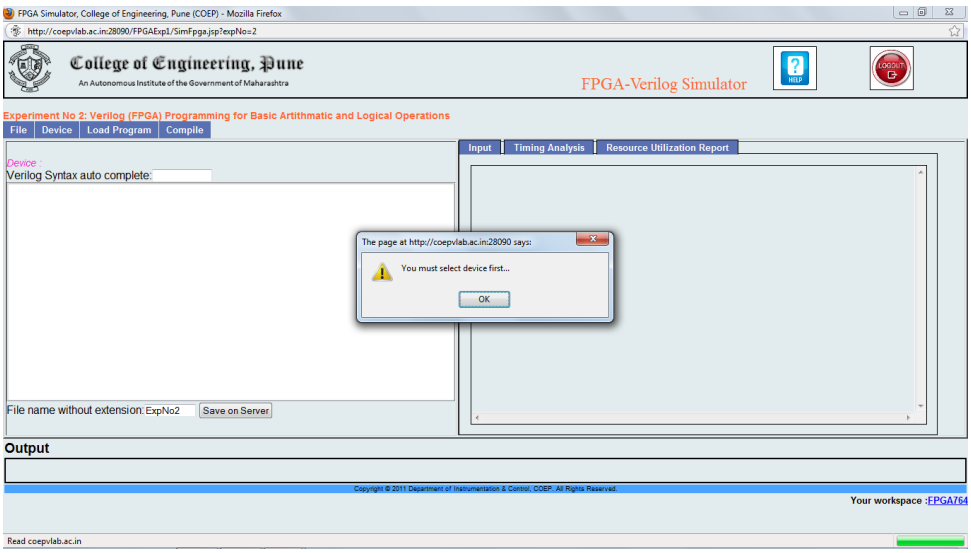


Figure 3 Front Panel of FPGA-Verilog Simulator

Input window: This is the section where user/programmer can select the input format and gives the value of input.

Editor window: This is the space where user/programmer can write, edit or load a Verilog program.

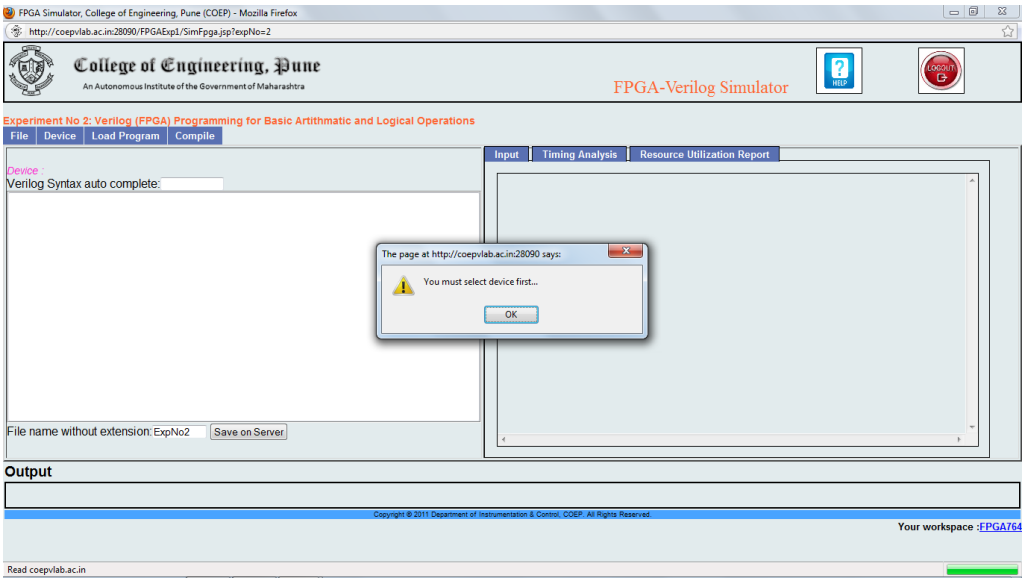
Output window: The result of compilation as well as output of Program appears in output window.

Simulation window: In this window shows simulated output of a Verilog program.

1.7.2 Using the Simulator

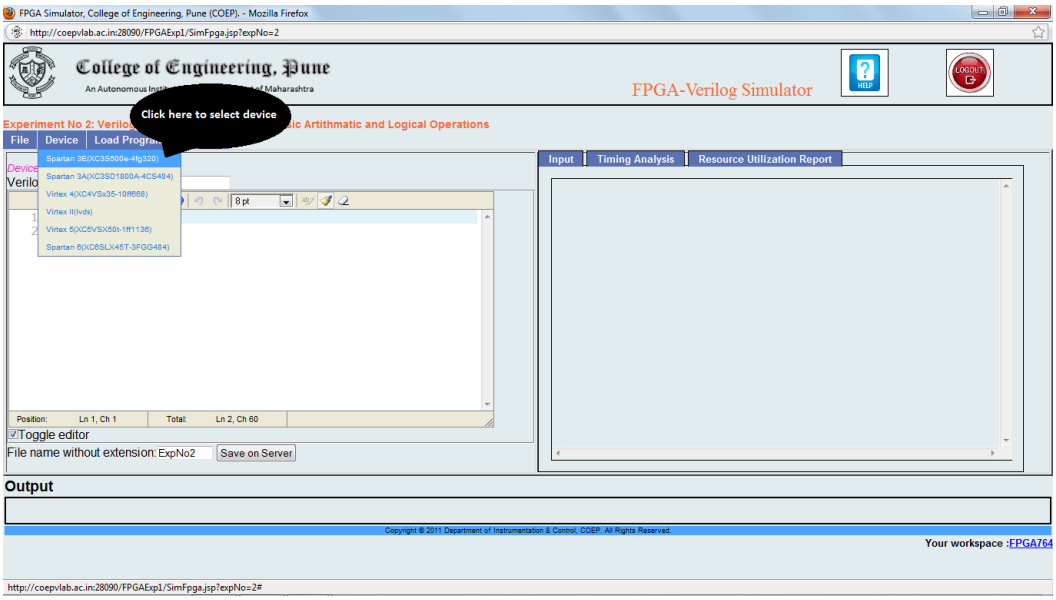
STEP 1

OPEN THE SIMULATOR



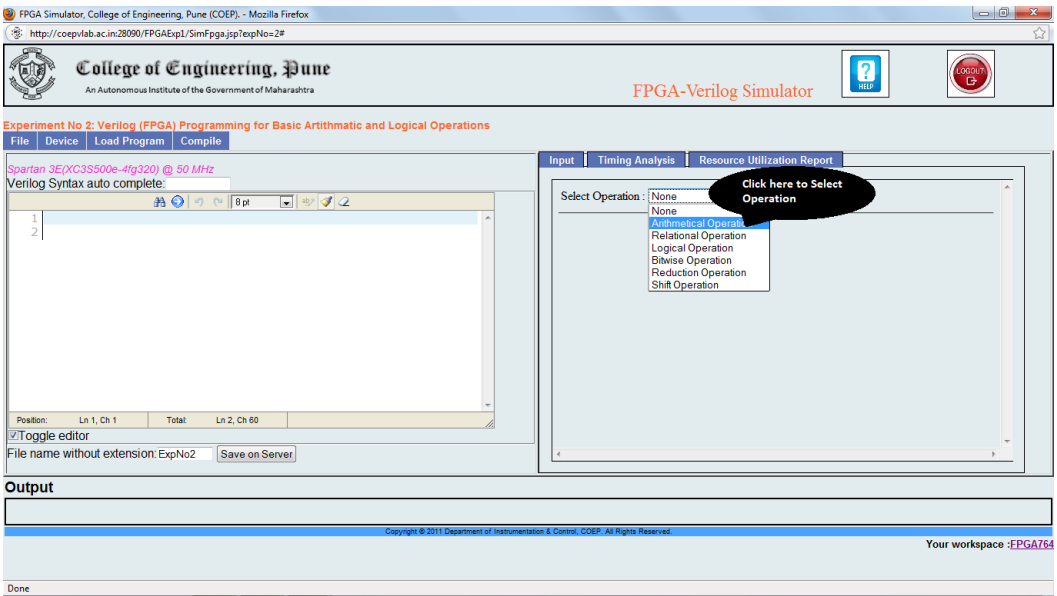
STEP 2

SELECT A DEVICE



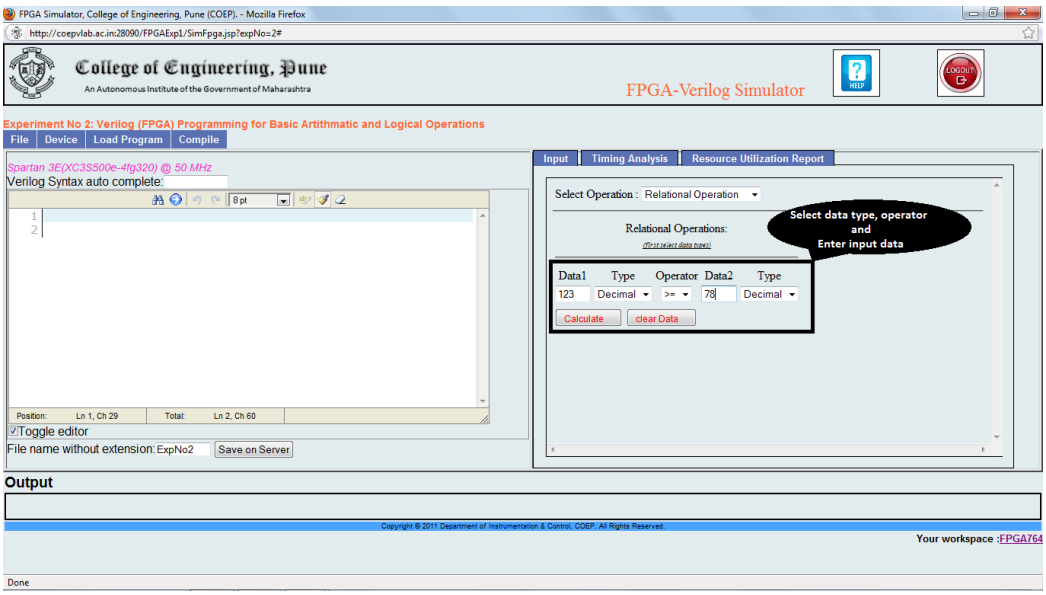
STEP 3

SELECT OPERATION



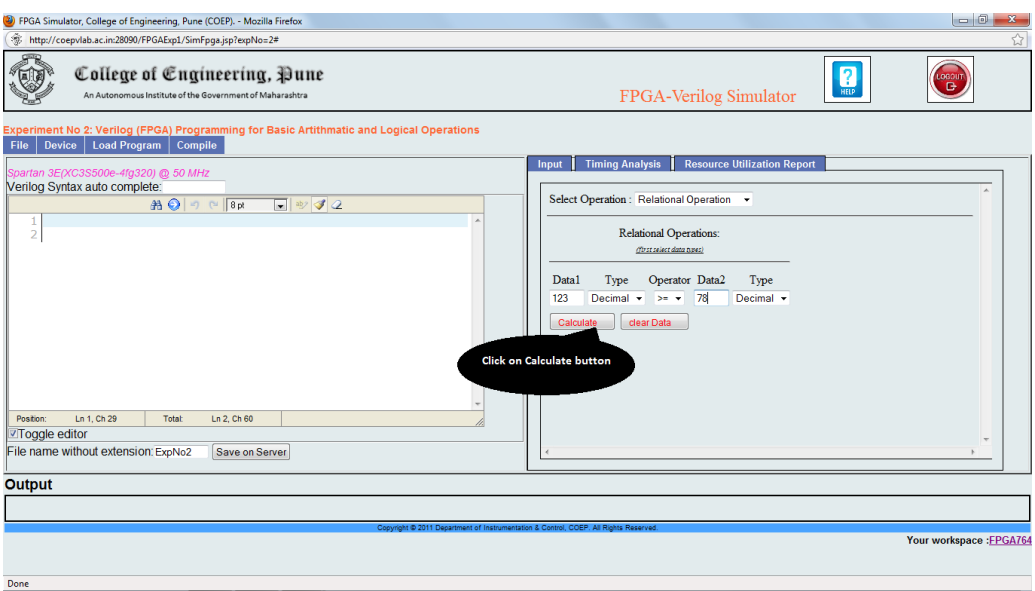
STEP 4

SELECT DATA TYPE, OPERATOR AND ENTER INPUT VALUE



STEP 5

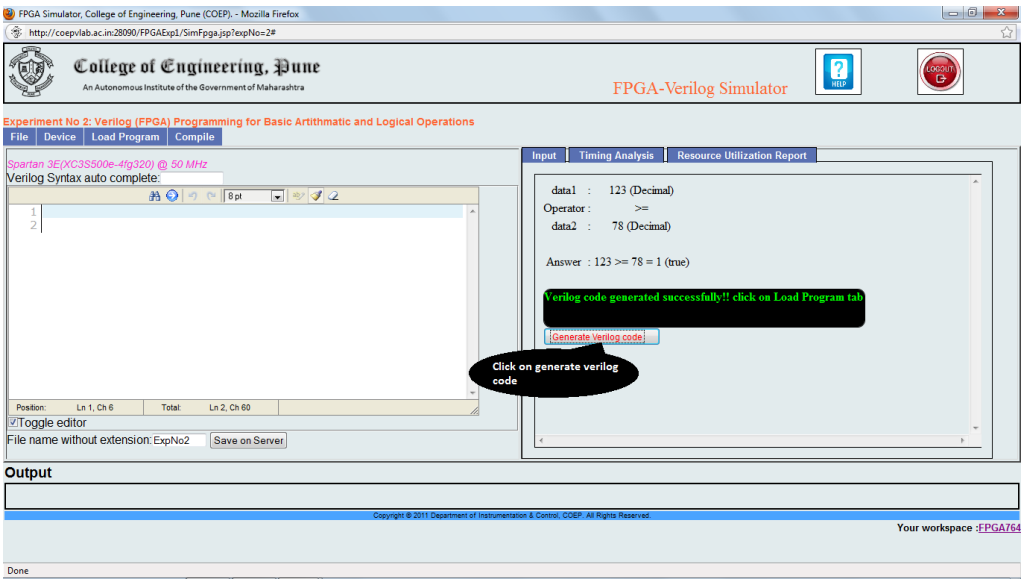
CLICK ON CALCULATE BUTTON



STEP 6

GENERATE

VERILOG CODE

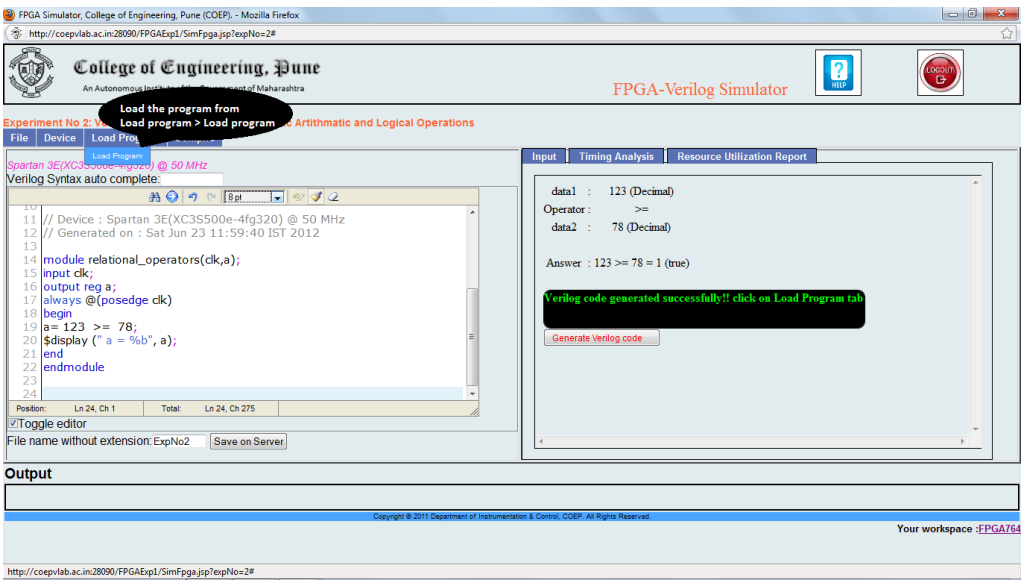


STEP 7

LOAD

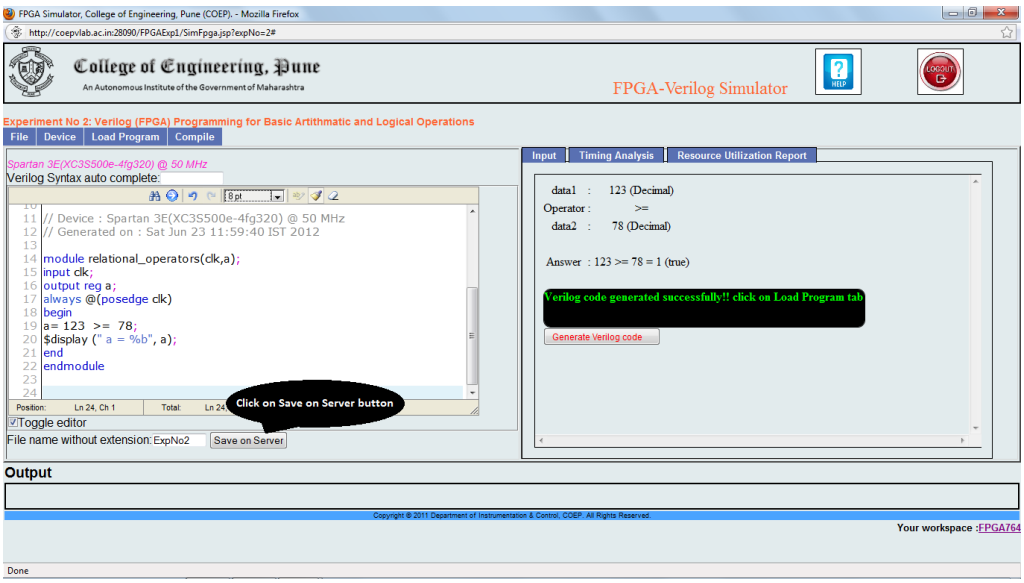
GENERATED

VERILOG CODE



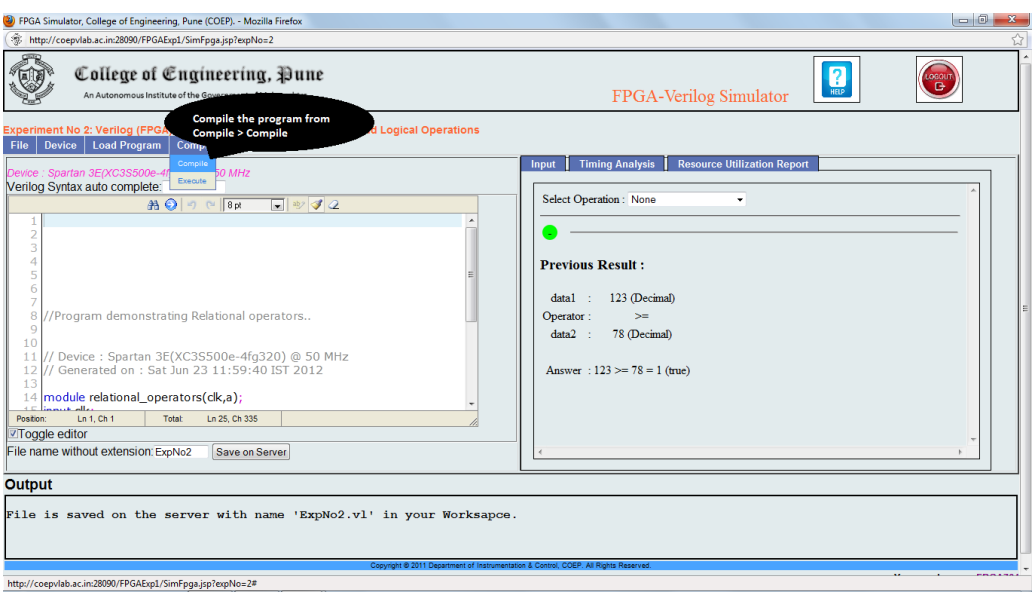
STEP 8

SAVE VERILOG CODE ON SERVER



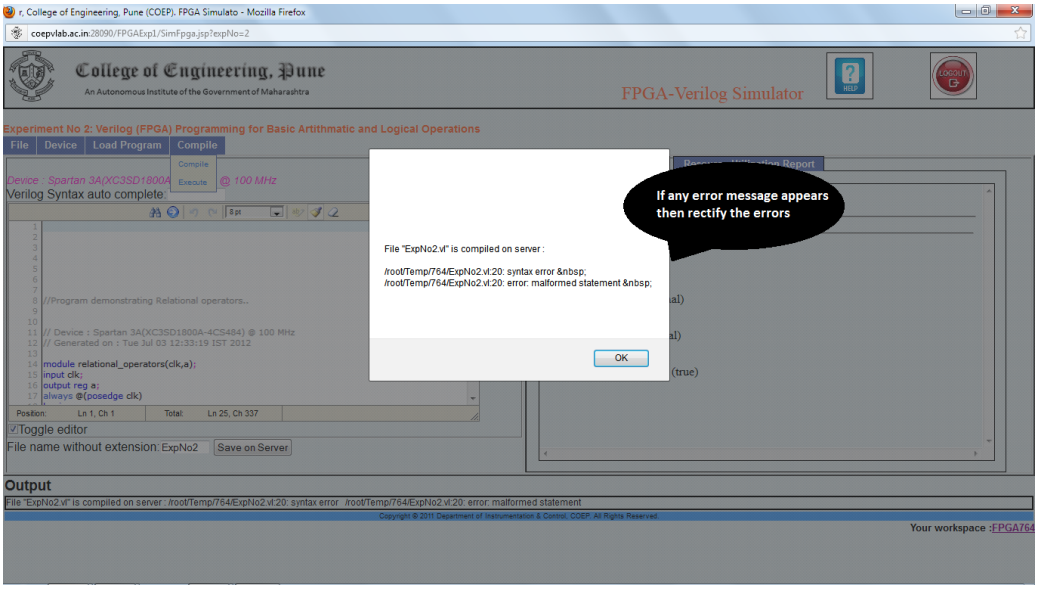
STEP 9

COMPILE THE SAVED VERILOG CODE



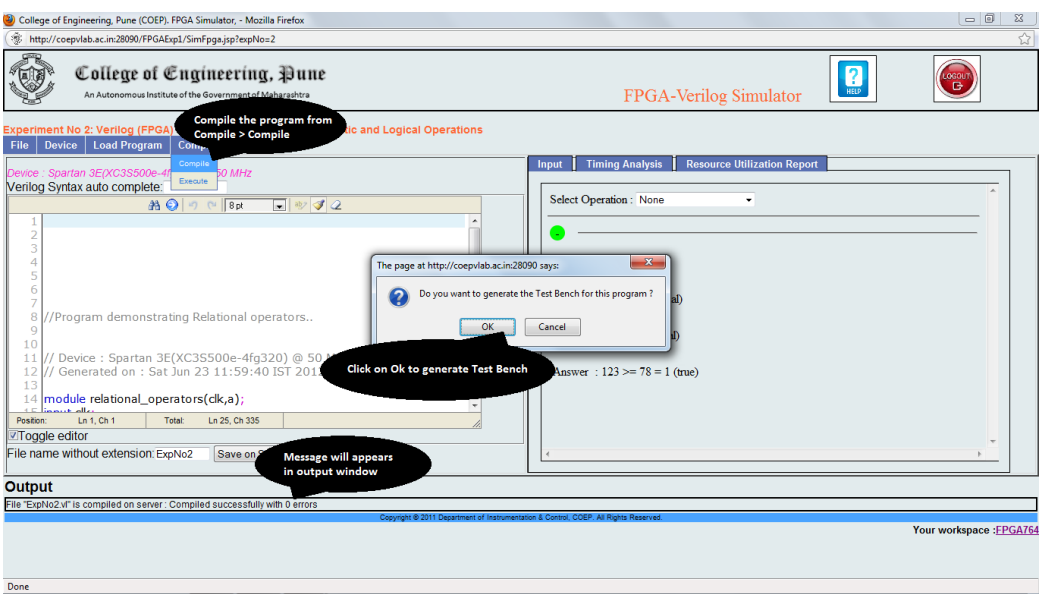
STEP 10

IF ANY ERROR IN THE VERILOG CODE



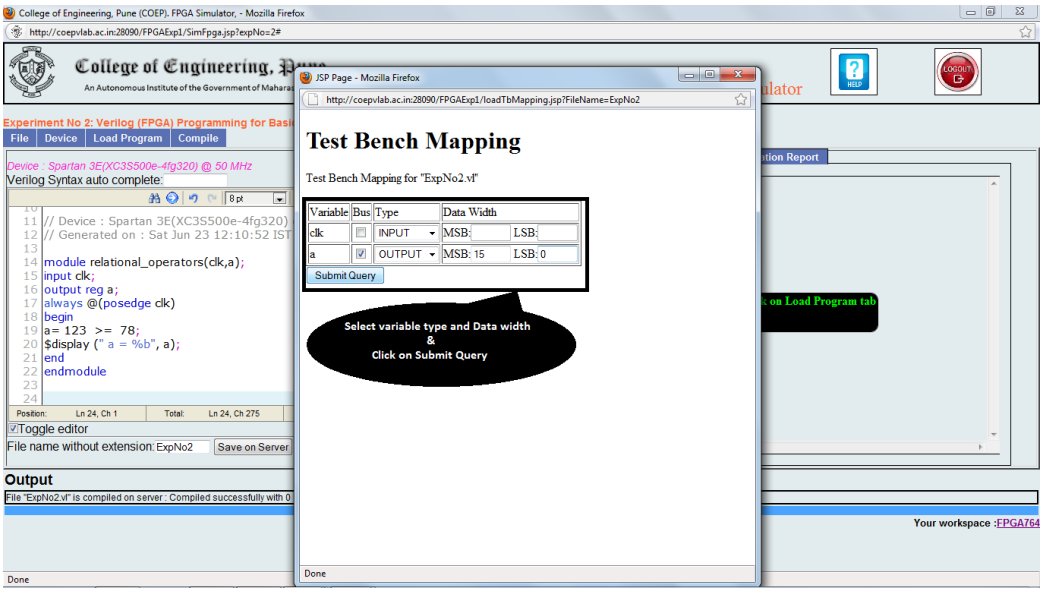
STEP 11

AFTER CORRECTING THE ERRORS SAVE THE PROGRAM AND AGAIN RECOMPILE AND GENERATE TEST BENCH



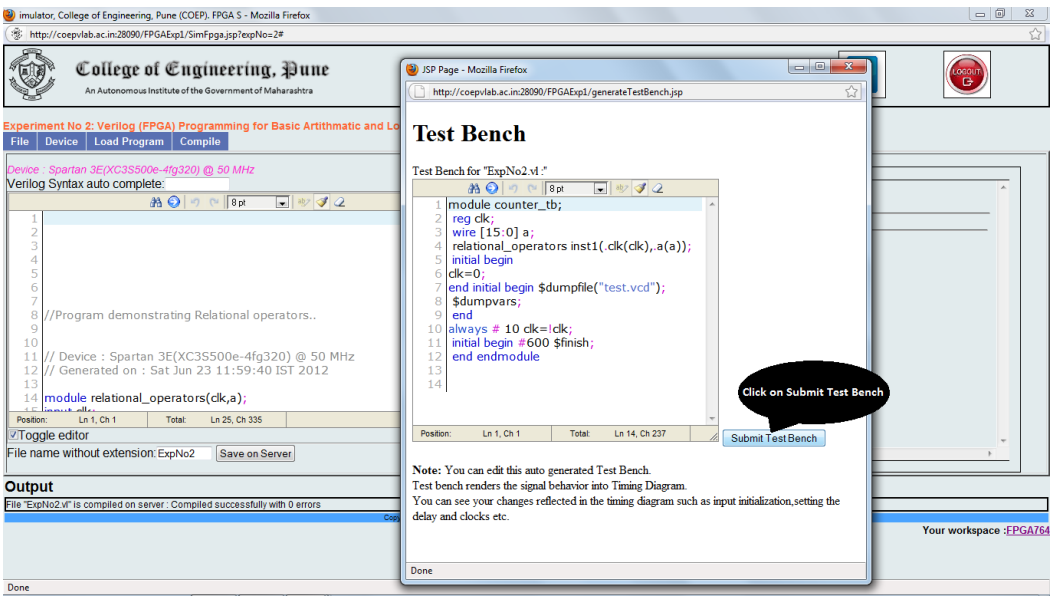
STEP 12

SELECT VARIABLE TYPE AND DATA WIDTH



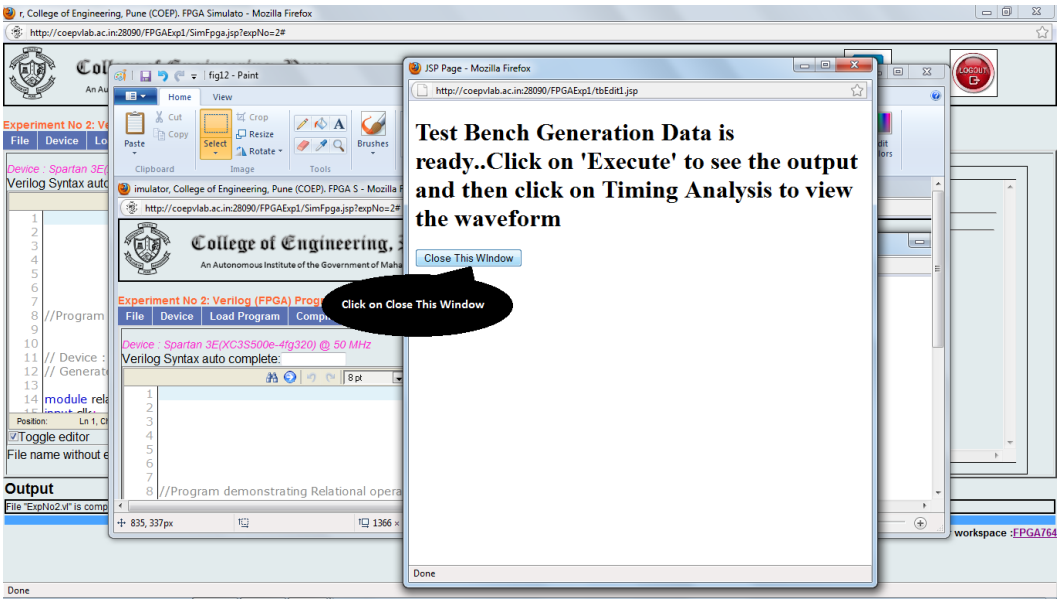
STEP 13

SUBMIT TEST BENCH



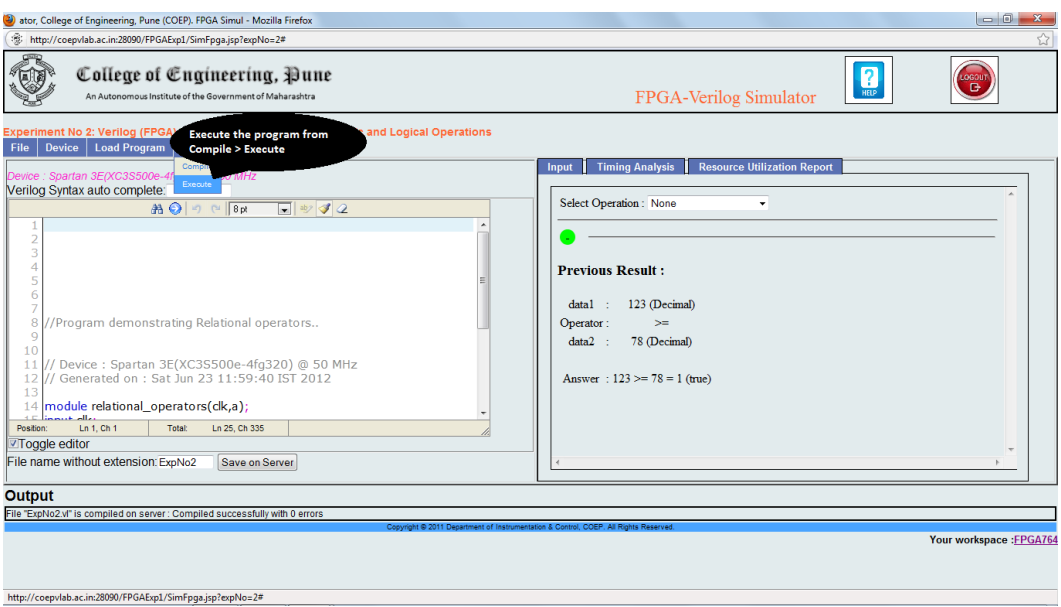
STEP 14

CLOSE THE TEST BENCH WINDOW



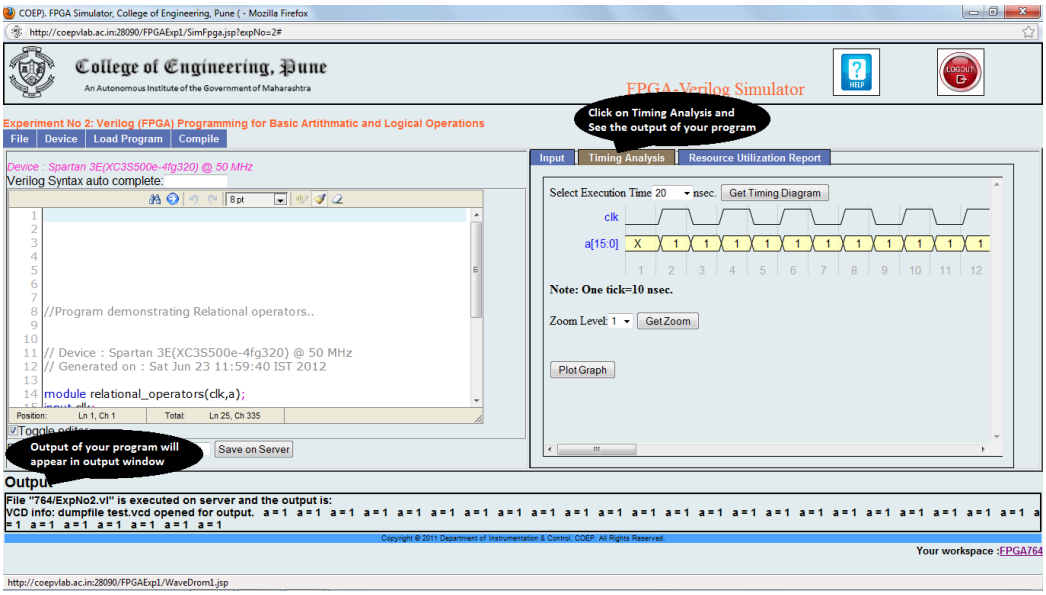
STEP 15

EXECUTE THE VERILOG CODE



STEP 16

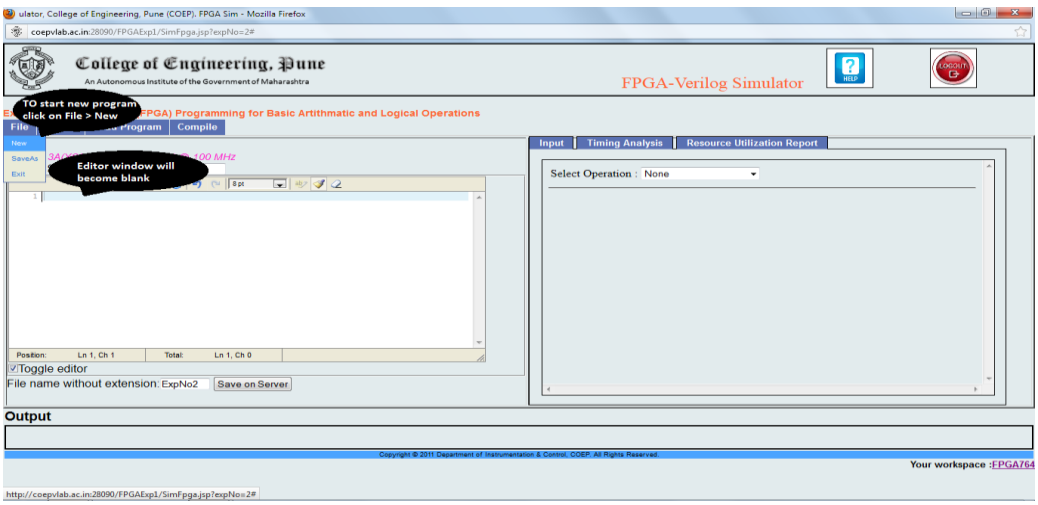
SEE THE OUTPUT OF VERILOG CODE IN TIMING ANALYSIS



1.8 FPGA-Verilog simulator's additional features

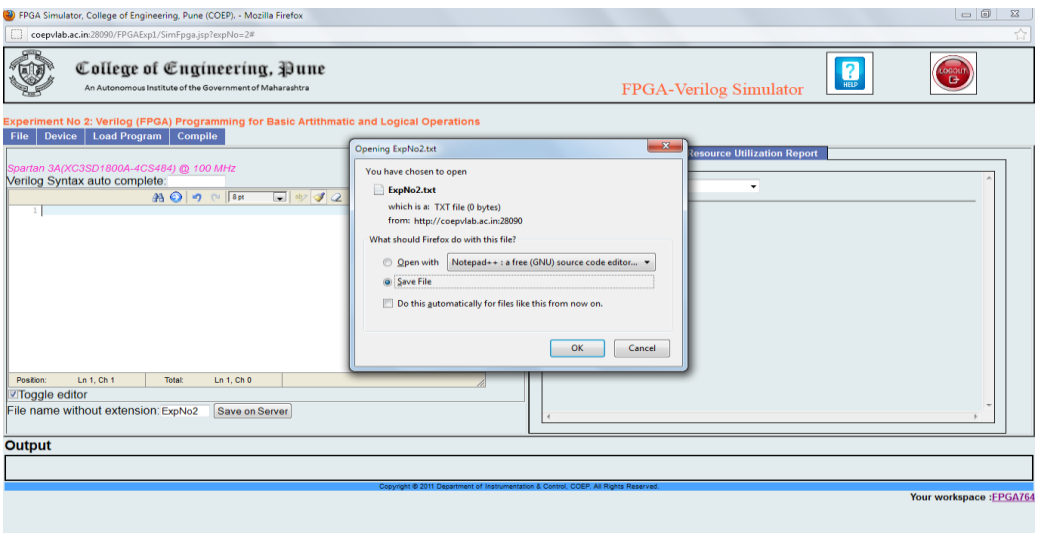
NEW FILE

This option enables user to create new file as a blank file in editor Window



SAVE AS

This option enables the user to save the program written in editor window at the desired location of his/her own PC.



1.9 Contact Information

1. DN Sonawane: *dns.instru@coep.ac.in* (Technical)

2. Kalyani Bhole: *kab.instru@coep.ac.in* (Technical)

3. Suchakra: *suchakra@gmail.com* (Documentation)

D.N. Sonawane
Department of Instrumentation and Control,
College of Engineering, Pune
Maharashtra, INDIA