DAILY ASSESSMENT FORMAT

Date:	**05_06_2020**	**Name:**	Poovi j gowda
Course:	**Digital design using verilog**	**USN:**	**4al17ec071**
Topic:	**VERILOG TUTORIALS AND FPGA**	**Semester & Section:**	**6th b**
Github Repository:	Poorvi-2000		

FORENOON SESSION DETAILS

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.

FPGA Projects on FPGA4student

Some of the FPGA projects can be FPGA tutorials such as What is FPGA Programming, image processing on FPGA, matrix multiplication on FPGA

Verilog Tutorial

By

Deepak Kumar Tala

http://www.asic-world.com

Page 1 / 227

Image of session

Report – Report can be typed or hand written for up to two pages.

VERILOG TUTORIALS

Verilog is a Hardware Description Language; a textual format for describing electronic circuits and syster for verification through simulation, for timing analysis, for test analysis (testability The Verilog HDL is a IEEE standard - number 1364. The first version of the IEEE standard for Verilog w this is the version use by most Verilog users. The IEEE Verilog standard document is known as the authoritative definition of th Verilog HDL.A further revision of the Verilog standard was published in System Verilog is a huge set c extensions to Verilog, and was first published as an IEEE standard in 200 System Verilog. IEEE Std 136 also defines the Programming Language Interface, or PLI. This is a collec between Verilog and other
Note that VHDL is not an abbreviation for Verilog HDL - Verilog and VHDL are two different HDLs.
The
The history of the Verilog HDL goes back to the 1980s, when a company called Gateway Design Automation developed a logic simulator, Verilog-XL, and with it a hardware description language. Cadence Design Systems acquired Gateway in 1989, and with it the rights to the language and the simulator. In 1990, Cadence put the language (but not the simulator) into the public domain, with the intention that it should become a standard, non-proprietary language. The Verilog HDL is now maintained by a non profit making organisation, Accellera, which was formed from the merger of Open Verilog International (OVI) and VHDL International. OVI had the task of taking the language through the IEEE standardisation procedure. In December 1995 Verilog HDL became IEEE Std. 1364-1995. A

a

significantly revised version was published in 2001: IEEE Std. 1364-2001. There was a further revision in 2005 but this only added a few minor changes. System level
Verilog is not ideally suited for abstract system-level simulation, prior to the hardware-software split. This is to some extent addressed by System Verilog. Unlike VHDL, which has support for user-defined types and overloaded operators which allow the designer to abstract his work into the domain of the problem, Verilog restricts the designer to working with pre-defined system functions and tasks for stochastic simulation and can be used for modelling performance, throughput and queueing but only in so far as those built-in langauge features allow. Designers occasionally use the stochastic level of abstraction for this phase of the design process. Digital
Verilog is suitable for use today in the digital hardware design process, from functional simulation, manual design and logic synthesis down to gate-level simulation. Verilog tools provide an integrated design environment in this area.
erilog is also suited for specialized implementation-level design verification tools such as fault simulation, switch level simulation and worst case timing simulation. Verilog can be used to simulate gate level fanout loading effects and routing delays through the import of SDF files. he RTL level of abstraction is used for functional simulation prior to synthesis. The gate level of abstraction exists post-synthesis but this level of abstraction is not often created by the designer, it is a level of abstraction adopted by the EDA tools. Analog
Because of Verilog's flexibility as a programming language, it has been stretched to handle analog simulation in limited cases. There is a draft standard – Verilog-AMS – that addresses analog and mixed signal simulation.

FPGA

FPGA Basics – A Look Under the Hood An introductory look inside Field Programmable Gate Arrays.
We'll go over: Strengths & Weaknesses of FPGAs How FPGAs work What's inside an FPGA So you keep hearing about FPGAs being utilized in more and more applications, but aren't sure whether it makes sense to switch to a new technology. Or maybe you're just getting into the embedded world and want to figure out if an FPGA-based system makes sense for you or not .This paper provides an overview of some of the key elements of FPGAs for engineers interested in utilizing FPGA-based technologies. It's worth noting that this is a complex topic, and as such, some topics are not covered, some are just introductory, and others will evolve over time. This paper should still give you a lot of helpful information if you're new to the world of FPGAs .What are the most important things you should know right away? Get out of the software mindset – You're not writing software. Let me say that again because this is the single most important point if you're thinking about working with FPGAs. You-are-NOT writing software .You're designing a digital circuit. You're using code to tell the chip how to configure itself. Plan for lots of bugs – yes, plan for them. They are going to happen. Way more than you expected. If you're a newbie developer, you need to pull in someone that has experience with FPGA development to help with this estimate .Application-specific realities – you ought to concern yourself with realities revolving around cyber security and safety, as FPGAs are a different animal than what you're likely used to. What is an FPGA? An FPGA is a (mostly) digital, (re-)configurable ASIC. I say mostly because there are analog and mixed-signal aspects to modern FPGAs. For example, some have A/D converters and PLLs. I put re- in parenthesis because there are actually one-time programmable FPGAs, where once you configure them, that's it, never again. However, most FPGAs you'll come across are going to be re-configurable. So what do I mean by digitally configurable ASIC?I mean that at the core of it, you're designing a digital logic circuit, as in AND, OR, NOT, flip-flops, etc. Of course that's not entirely accurate and there's much more to it than that, but that is the gist at its core. he players –There are currently two big boys: Altera (part of Intel) and Xilinx, and some supporting players (e.g.

Actel (owned by Microsemi)).The main underlying technology options are SRAM-based (this is the

most common technology), flash, and anti-fuse. As you might imagine, each option has its own pros and cons. Check this out for some more details. Strengths / best suited for: Much of what will make it worthwhile to utilize an FPGA comes down to the low-level functions being performed within the device. There are four processing/algorithm attributes defined below that FPGAs are generally well suited for. While just one of these needs may drive you toward an FPGA, the more of these your application has, the more an FPGA-based solution will appeal. Parallel processes – if you need to process several input channels of information (e.g. many simultaneous A/D channels) or control several channels at once (e.g. several PID loops). High data-to-clock-rate-ratio – if you've got lots of calculations that need to be executed over and over and over again, essentially continuously. The advantage is that you're not tying up a centralized processor. Each function can operate on its own. Large quantities of deterministic I/O – the amount of determinism that you can achieve with an FPGA will usually far surpass that of a typical sequential processor. If there are too many operations within your required loop rate on a sequential processor, you may not even have enough time to close the loop to update all of the I/O within the allotted time. Signal processing – includes algorithms such as digital filtering, demodulation, detection algorithms, frequency domain processing, image processing, or control algorithms. Weaknesses / not optimal for: With any significant benefit, there's often times a corresponding cost. In the case of FPGAs, the following are generally the main disadvantages of FPGA based solutions. Complex calculations infrequently – If the majority of your algorithms only need to make a computation less than 1% of the time, you've generally still allocated those logic resources for a particular function (there are exceptions to this), so they're still sitting there on your FPGA, not doing anything useful for a significant amount of time. Sorting/searching – this really falls into the category of a sequential process. There are algorithms that attempt to reduce the number of computations involved, but in general, this is a sequential process that doesn't easily lend itself to efficient use of parallel logical resources. Check out the sorting section here and check out this article
here for some more info .Floating point arithmetic – historically, the basic arithmetic elements within an FPGA have been fixed-point binary elements at their core. In some cases, floating point math can be achieved (see Xilinx FP Operator and Altera FP White Paper), but it will chew up a lot of logical resources. Be mindful of single-precision vs double-precision, as well as deviations from standards. However, this FPGA weakness appears to be starting to fade, as hardened floating-point DSP blocks are starting to be embedded within some FPGAs (see Altera Arria 10 Hard Floating Point DSP Block).Very low power – Some FPGAs have low power modes (hibernate and/or suspend) to help reduce current consumption, and some may require external mode control ICs to get the most out of this. Check out an example low power mode FPGA here. There are both static and dynamic aspects to power consumption. Check out these power estimation spreadsheets to start to get a sense of power utilization under various conditions. However, if low power is critical, you can generally do better power-wise with low-power architected microprocessors or microcontrollers. Very low cost – while FPGA costs have come down drastically over the last decade or so, they are still generally more expensive than sequential processors. How Does an FPGA work? You're designing a digital circuit more than anything else, basically at one layer of abstraction above the logic gate (AND, OR, NOT) level. At the most basic level, you need to think about how you're specifying the layout and equations at the level of LUTs (Look-Up Tables) and FFs (Flip-Flops).Otherwise you're circuit can get very large and slow very quickly. You've got a very detailed level of control at your fingertips, which is very powerful, but can be overwhelming, so start slow. You'll be determining the # of bits, and exact math / structure of each function. An FPGA is a synchronous device, meaning that logical operations are performed on a clock cycle-by-cycle basis. Flip-flops are the core element to enabling this structure .In general, you're going to put digital data into an FPGA and get digital data out of it through various low voltage digital I/O lines, sometimes many bits in parallel (maybe through one or more A/D converter outputs or an external DRAM chip), sometimes through high-speed

serial I/O (maybe connecting to an Ethernet PHY or USB chip).What’s inside – Core components (or at least what everyone likes to think about):LUT (Look-Up Table) –The name

LUT in the context of FPGAs is actually misleading, as it doesn't convey the full power of this logical resource. The obvious use of a LUT is as a logic lookup table (see examples here and here), generally with 4 to 6 inputs and 1 to 2 outputs to specify any logical operation that fits within those bounds. There are however two other common uses for a LUT:LUT as a shift register – shift registers are very useful for things like delaying the timing of an operation to align the outputs of one algorithm with another. Size varies based on FPGA.LUT as a small memory – you can configure the LUT logic as a VERY small volatile random-access memory block. Size varies based on FPGAFF (Flip-flop) –Flip-flops store the output of a combinational logic calculation. This is a critical element in FPGA design because you can only allow so much asynchronous logic and routing to occur before it is registered by a synchronous resource (the flipflop), otherwise the FPGA won't make timing. It's the core of how an FPGA works. Flip-flops can be used to register data every clock cycle, latch data, gate off data, or enable signals .Block Memory – It's important to note that there are generally several types of memory in an FPGA. We mentioned the configuration of a LUT resource. Another is essentially program memory, which is intended to store the compiled version of the FPGA program itself (this may be part of the FPGA chip or as a separate non-volatile memory chip). What we're referring to here though, is neither of those types of memory. Here we're referring to dedicated blocks of volatile user memory within the FPGA. This memory block is generally on the order of thousands of bits of memory, is configurable in width and depth, and multiple blocks of memory can be chained together to create larger memory elements. They can generally be configured as either single-port or dual-port random access, or as a FIFO. There will generally be many block memory elements within an FPGA .Multipliers or DSP blocks –Have you ever seen the number of digital logic resources that it takes to create a 16-bit by 16-bit multiplier? It's pretty crazy, and would chew through your logical and routing resources pretty quickly. Check out the 2-bit by 2-bit FPGA vendors solve this problem with dedicated silicon to lay down something on the order of 18-bit multiplier blocks. Some architectures have recognized the utility of digital signal processing taking place, and have taken it a step further with dedicated DSP (Digital Signal Processing) blocks, which can not only multiply, but add and accumulate as well.I/O

(Input/Output) –If you're going to do something useful with an FPGA, you generally have to get data from and/or provide data outside the FPGA. To facilitate this, FPGAs will include I/O blocks that allow for various voltage standards (e.g. LVCMOS, LVDS) as well as timing delay elements to help align multiple signals with one another (e.g. for a parallel bus to an external RAM chip).Clocking and routing –This is really a more advanced topic, but critical enough to at least introduce. You'll likely use an external oscillator and feed it into clocking resources that can multiply, divide, and provide phaseshifted versions of your clock to various parts of the FPGA.Routing resources not only route your clock to various parts of the FPGA, but also your data. Routing resources within an FPGA are one of the most underappreciated elements, but so critical. Check out this sea of madness:What's Inside – Advanced componentsHard cores – These are functional blocks that (at least for the most part) have their own dedicated logical resources. In other words, they are already embedded into your FPGA silicon. You configure them with various parameters and tell the tools to enable them for you. This could include functions such as high-speed communications (e.g. high-speed serial, Ethernet), low-speed A/D converters for things like measuring slowly varying voltages, and microprocessor cores to handle some of the functions that FPGA logic is not as well suited for.Soft cores – These are functional blocks that don't have their own dedicated logical resources. In other words, they are laid out with your core logic resources. You configure them with various parameters and tell the tools to build them for you. This could include everything from DDR memory interfaces to FFT cores to FIR filters to microprocessors to CORDICs. The library of available soft cores can be impressive.

T-FLIP FLOP

```
module tff
    (t,clk,q,qb); input t,clk;
```

```
output q,qb;
reg q, qb;
initial
begin
q=0;
q=1; end
always@(posedge (clk)) begin
if(t==0)
q=q; else
q=qb;
qb=~q; end
endmodule
```

Course:	**python**	**USN:**	4al17ec071
Topic:	**BUILD A DATA COLLECTOR WEB APP WITH POST GRESQL AND FLASK**	**Semester & Section:**	**6th b**

AFTERNOON SESSION DETAILS

Image of session

Report – Report can be typed or hand written for up to two pages.

Flask start up and configuration

Like 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.