**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **14-07-2020** | **Name:** | **Karthik J** |
| **Course:** | Introduction to FPGA Design for Embedded Systems | **USN:** | **4AL16EC030** |
| **Topic:** | Week 2-Design Flow in FPGA | **Semester & Section:** | **8TH A** |
| **GitHub Repository:** | Karthik-J |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| A Field Programmable Gate Array, or FPGA, is a semiconductor device that comprises of logic blocks which are programmed to execute a specific set of functions. These programmable logic blocks are connected to each other with the help of an interconnect matrix. These interconnects are responsible for connecting the logic blocks and facilitating the flow of signals across the chip. This structure is arranged in the form of a two-dimensional array consisting of logic blocks, interconnects, and I/O blocks that connect it with the input and output signals. A logic block itself is composed of a look up table or LUT and a flip flop or FF and a multiplexer.    One of the major benefits of using FPGAs is that they are reprogrammable, meaning they can be modified to serve a function that is completely different from the one they were performing before every time the designer uploads a new code to the FPGA.   **FPGA Design Flow**   The **FPGA design flow** comprises of several different steps or phases, including design entry, synthesis, implementation, and device programming. We will explore each of these phases in detail.       ****Design Entry****   Design entry can be done using various techniques, such as schematics, through Hardware Description Language or HDL, or you may even combine the two and use a best of both worlds approach using tools that can convert HDL into schematics and vice versa depending on your FPGA design and preference. Generally, for a design that deals more with complex systems, it is better to opt for HDL, a quicker, language-based process that rids you of the need to design in lower level hardware, while schematics is a good choice for someone who wishes to design hardware because it gives more visibility to the entire system. You can also opt to go for a state-machines based approach, but it is largely limited and unused currently. It is suited for designers who view their design as a series of states.    There are benefits and downsides associated with each approach. While a schematic based technique is easier to read and comprehend, it tends to only work with relatively smaller projects. HDL based approaches, on the other hand, tend to be fast and easy to implement, and today is most popular design entry for FPGA designs.   ****Synthesis****   After the design has been entered in the form of code, this phase is where it is translated into an actual circuit with elements such as gates, flip flops, and multipliers among others. Your input HDL is essentially converted into a netlist which lists the logic elements you will be needing for your project and the interconnects needed in the specific hierarchy.    The process begins with a syntax check once you feed in your HDL based design. It is then optimized by the reduction of logic, elimination of redundant logic, and the reduction of the size of the design while simultaneously making it faster to implement. The last step is to map out the technology by connecting the design to the logic, estimating the associated time, and churning out the design netlists which are subsequently saved.  FPGA synthesis is performed by dedicated synthesis tools. Cadence, Synopsys and Mentor Graphics are EDA companies that develop, sell and market FPGA synthesis tools.   ****Implementation****   This phase is where the layout of your design will be determined and consists of three steps: translate, map, and place & route. The tools used in this step are provided by the FPGA vendors because they know best how to translate a synthesized netlist into an FPGA.    The first step for the tools is to gather all the constraints that are set by the user together with the netlist files. These constraints can be regarding the assignment and position of the pins, the requirements regarding timing such as the maximum delay or the input period of the clock.    Then the tool maps out the implementation by comparing the resource requirement specified in the files to the resources actually available on the FPGA being used. The circuit is divided into the logic blocks or elements in the form of sub blocks. As a result, your entire design is placed in specific logic blocks and is ‘mapped out’ into the FPGA.    The next step is to connect and route all the signals accordance with the constraints set by the user between all the logic blocks and IO blocks.   ****Device Programming****   The last step in the process is to finally load the mapped out and completely routed design into the FPGA. For that reason, you will need a to generate a BitSteam file.   **FPGA Verification & Simulation**   At the end of each step in the FPGA design flow, you have the opportunity to simulate and test you design. There are essentially 3 points allowed by the FPGA design flow: at design entry, post synthesis, or post implementation.        It is needless to mention that verifying that the implemented design performs the required funtionality an important part of the FPGA design flow.   ****Behavioral Simulation (At Design Entry)****   Behavioral simulation, called also RTL simulation, is performed before synthesis. This fast simulation can be used to check the functionality of the design without constraints. Use this simulation frequently to test your code and find logic errors.   ****Functional Simulation (Post Synthesis)****   The functionality the design can be verified using functional simulation after the synthesis process has completed. It is a netlist level simulation that ignore timing related issues.   ****Timing Simulation (At Implementation)****   This simulation will give you the most accurate picture of your design behavior.  It takes into account the target FPGA chip and all the logic blocks functionality, wiring, delays and much more.  Timing simulation takes longer time and provides much more details than the previous simulation. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Date:** | | **14-07-2020** | **Name:** | **Karthik J** |  |
| **Course:** | | Trail Head Sales force Developer | **USN:** | **4AL16EC030** |  |
| **Topic:** | | Platform | **Semester & Section:** | **8th A** |  |
|  | **AFTERNOON SESSION DETAILS** | | | | |
|  | **Image of session**    At Salesforce, we group our services by clouds. There’s Sales Cloud for CRM,  Service Cloud for customer support, and a handful of other clouds that help  companies support their business functions. And while each of these clouds serves  a unique purpose, there’s one thing they all have in common: the power of the  Salesforce platform.  What is the Salesforce platform, exactly?  Like any platform, the Salesforce platform is a group of technologies that supports  the development of other technologies on top of it. What makes it unique is that the  platform supports not only all the Salesforce clouds, but it also supports custom  functionality built by our customers and partners. This functionality ranges from  simple page layouts to full-scale applications.  If you’re here today, we’re assuming you know a bit about software development.  Throughout this module, we’re going to give you an overview of development on  the Salesforce platform. We talk about some of the pillars of Salesforce  development and how they work together to create a robust system. We even touch  on some common questions that developers new to the platform run into as they get  started.  Before we continue, let’s make sure we’re on the same page. If you’re brand new to  Salesforce and you haven’t completed the  Salesforce Platform Basics module  , we  suggest you do that before you keep reading.  Once you’re done with that, you’re ready to get started!  Platform Building Blocks  As we mentioned, the platform not only forms the foundation of core Salesforce  products like Sales Cloud and Service Cloud, but it also lets you build your own  functionality. Building your own functionality can mean customizing existing  Salesforce offerings or it can mean building something from scratch.  Let’s focus on that latter part and talk about what the Salesforce platform offers  developers.  Our core platform lets you develop custom data models and applications for  desktop and mobile. And with the platform behind your development, you can build  robust systems at a rapid pace.  And then there’s the Heroku platform. Heroku gives developers the power to build  highly scalable web apps and back-end services using Python, Ruby, Go, and more.  It also provides database tools to sync seamlessly with data from Salesforce.  And then there’s the host of Salesforce APIs. These let developers integrate and  connect all their enterprise data, networks, and identity information.  And then there’s the Mobile SDK. The Mobile SDK is a suite of technologies that  lets you build native, HTML5, and hybrid apps that have the same reliability and  security as the Salesforce app.  And then... wait. Let’s stop for a second.  The problem with the platform and all its parts is that listing them out takes a really  long time. And just talking about them doesn’t help you understand everything they  do. Let’s take a different approach and talk about what we can do with the platform.  Or, more precisely, what we can build with it.  The DreamHouse App  Let’s float a scenario. Throughout the rest of this module, we use this scenario to  explore the many exciting tools and technologies that the Salesforce platform  provides.  You’re a developer for DreamHouse Realty, a company that aggregates real estate  listings to better connect homebuyers and real estate agents. Your boss asks you to  build a new system to track real estate listings. Your internal employees will use it  to track and communicate about properties. Your partner real estate brokers will use  it to access information about customers. And your customers will view properties  and contact brokers for viewings.  Building an app like this one from scratch isn’t an easy thing to do. Taking on this  project in real life can involve a long, complicated list of functional requirements  and the implementation of special integrations for your company’s business data.  Working by yourself, it can take you months to get something out the door.  But before your stress builds and you melt into a puddle of existential dread,  remember: You’ve got the platform. And building complex business applications at  a breakneck pace is what the platform’s all about.  We’re going to show you a fully functional version of the DreamHouse app so you  can get a feel for how it was built. As we move through, we discuss important  Salesforce development concepts using the app to guide us.  Install the DreamHouse App  To follow along and practice the steps in this module, you need to install the  DreamHouse package in your Trailhead Playground. Follow the instructions here to  launch a playground and install the package. You also use this package and  playground when it’s time to complete the hands-on challenge.  Launch your Trailhead Playground by scrolling to the bottom of this page and  clicking  Launch  . If you see a tab in your org labeled Install a Package, great.  Follow the steps below.  If not, from the App Launcher (  ), find and select  Playground Starter  and follow  the steps. If you don’t see the Playground Starter app, copy  this package installation  link  and check out  Install a Package or App to Complete a Trailhead Challenge  on  Trailhead Help.  1.  Click the Install a Package tab.  2.  Paste  04tB00000009UeX  into the field.  3.  Click  Install | | | | |
|  |  | | | | |