**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **13-07-2020** | **Name:** | **Karthik J** |
| **Course:** | Introduction to FPGA Design for Embedded Systems | **USN:** | **4AL16EC030** |
| **Topic:** | Week 1 | **Semester & Section:** | **8TH A** |
| **GitHub Repository:** | Karthik-J |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| FPGA  FPGA stands for field-programmable gate array. At its core, an FPGA is an array of interconnected digital subcircuits that implement common functions while also offering very high levels of flexibility. But getting a full picture of what an FPGA is requires more nuance. This article introduces the concepts behind FPGAs and briefly discuss what logic gates are, how to program an FPGA, and what makes an FPGA different from a microprocessor in design. What Is a Field-Programmable Gate Array? A good name can be quite informative, and I would consider “field-programmable gate array” to be a fairly good name. An FPGA is an **array of logic gates** (well, sort of—see below), and this array can be **programmed** (actually, “configured” is probably a better word) **in the field**, i.e., by the user of the device as opposed to the people who designed it. Let’s take a closer look at these essential characteristics.  [Logic gates](https://www.allaboutcircuits.com/textbook/digital/chpt-3/digital-signals-gates/) (AND, OR, XOR, etc.) are the basic building blocks of digital circuitry. It’s not surprising, then, that a digital device that is intended to be highly configurable (that is, "field-programmable") would consist of numerous gates that can be interconnected in a customizable way.  However, an FPGA is not a vast collection of individual Boolean gates. This would be a very suboptimal way to provide configurable-logic functionality because it would not take advantage of the fact that common operations can be implemented much more efficiently as fixed modules. The same principle is evident in the world of discrete digital ICs. You can buy ICs that consist of AND gates, OR gates, and so forth—but you wouldn’t want to build a shift register out of individual gates. Instead, you would buy a shift register IC.  An FPGA, then, is much more than an array of gates. It’s an array of carefully designed and interconnected digital subcircuits that efficiently implement common functions while also offering very high levels of flexibility. The digital subcircuits are called configurable logic blocks (CLBs), and they form the core of the FPGA’s programmable-logic capabilities  The CLBs need to interact with one another and with external circuitry. For these purposes, the FPGA uses a matrix of programmable interconnects and input/output (I/O) blocks. The FPGA’s “program” is stored in SRAM cells that influence the functionality of the CLBs and control the switches that establish the connection pathways.  A detailed explanation of a CLB’s internal structure and operation would require an entire article (if not multiple articles). The general idea is that CLBs include [look-up tables](https://www.allaboutcircuits.com/technical-articles/purpose-and-internal-functionality-of-fpga-look-up-tables/), storage elements (flip-flops or registers), and multiplexers that allow the CLB to perform Boolean, data-storage, and arithmetic operations.  An I/O block consists of various components that facilitate communication between the CLBs and other components on the board. These include pull-up/pull-down resistors, buffers, and inverters. Field-Programmable Logic How do we go about turning an array of CLBs into a digital circuit that does precisely what we want it to? At first glance, it seems like a rather complicated task. Indeed, FPGA implementation is generally considered more difficult than programming a microcontroller. However, FPGA development does not require thorough knowledge of CLB functionality or painstaking arrangement of internal interconnects, just as microcontroller development does not require thorough knowledge of a processor’s assembly-language instructions or internal control signals.  Actually, it is somewhat misleading to present an FPGA as a standalone component. FPGAs are always supported by development software that carries out the complicated process of converting a hardware design into the programming bits that determine the behavior of interconnects and CLBs. **Programmable logic array** LAs became available in 1975 to address the limitations imposed by the PROM architecture, where both planes AND and OR arrays were programmable. Logically, a PLA is a circuit that allows implementing Boolean functions in sum-of-product form. The number of AND functions in the AND array is independent of the number of inputs. Additional ANDs can be formed by simply introducing more rows into the array. Similarly, the number of OR functions in the OR array is independent of both the number of inputs and number of AND functions in the AND array. ORs can be formed by introducing additional columns into the array. Each place in the AND-matrix holds a small diode. Depending on the programming data, this diode is left unconnected, or connected to its input-line and product-line. While the unconnected diode will do nothing, the product-term line will be driven low by the connected diode whenever the corresponding input-line is low. This is the wired-AND operation: a product term will only remain high when none of the (connected!) input-lines is driven low.  PLA shares single product term across multiple ORs array, so highest logic density is available to the user. The number of fuse count is high in PALs. The main advantage of the PLA structure is that a very compact and space-efficient realization is possible in NMOS technology. Small self-conducting (enhancement-mode) NMOS transistors are used for the pull-up resistors, while a depletion-mode NMOS transistor is placed at each location in the AND- and OR-matrices.  Both **Programmable Array Logic** and **Programmable Logic Array** are types of PLDs (programmable logic devices), and these are mainly used for designing combination logic mutually by sequential logic. The main difference among these two is that PAL can be designed with a collection of AND gates and fixed collection of OR gates whereas PLA can be designed with a programmable array of AND although a fixed collection of OR gate. A programmable logic device offers a simple as well as flexible logic circuit designing. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Date:** | | **13-07-2020** | **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 | | | | |
|  |  | | | | |