**AILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **01-June-2020** | **Name:** | **Raziya Banu** |
| **Course:** | **HDL** | **USN:** | **4AL16EC058** |
| **Topic:** | **FPGA and its industry applications** | **Semester & Section:** | **8th sem & ‘B’ section** |
| **Github Repository:** |  |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report –**  In my first session today I have studied about the FPGA and its industry applications. Field-programmable gate arrayFrom Wikipedia, the free encyclopedia[Jump to navigation](https://en.wikipedia.org/wiki/Field-programmable_gate_array#mw-head)[Jump to search](https://en.wikipedia.org/wiki/Field-programmable_gate_array#p-search)"FPGA" redirects here. It is not to be confused with [Flip-chip pin grid array](https://en.wikipedia.org/wiki/Flip-chip_pin_grid_array). [https://upload.wikimedia.org/wikipedia/commons/thumb/f/fa/Altera_StratixIVGX_FPGA.jpg/220px-Altera_StratixIVGX_FPGA.jpg](https://en.wikipedia.org/wiki/File:Altera_StratixIVGX_FPGA.jpg)  **A [Stratix IV](https://en.wikipedia.org/wiki/Stratix_(FPGA)" \o "Stratix (FPGA)) FPGA from**[**Altera**](https://en.wikipedia.org/wiki/Altera)  A field-programmable gate array (FPGA) is an [integrated circuit](https://en.wikipedia.org/wiki/Integrated_circuit) designed to be configured by a customer or a designer after manufacturing – hence the term "[field-programmable](https://en.wikipedia.org/wiki/Field-programmability)". The FPGA configuration is generally specified using a [hardware description language](https://en.wikipedia.org/wiki/Hardware_description_language) (HDL), similar to that used for an [application-specific integrated circuit](https://en.wikipedia.org/wiki/Application-Specific_Integrated_Circuit) (ASIC). [Circuit diagrams](https://en.wikipedia.org/wiki/Circuit_diagram) were previously used to specify the configuration, but this is increasingly rare due to the advent of [electronic design automation](https://en.wikipedia.org/wiki/Electronic_design_automation) tools.  [https://upload.wikimedia.org/wikipedia/commons/thumb/a/a1/Xerox_ColorQube_8570_-_Main_controller_-_Xilinx_Spartan_XC3S400A-0205.jpg/220px-Xerox_ColorQube_8570_-_Main_controller_-_Xilinx_Spartan_XC3S400A-0205.jpg](https://en.wikipedia.org/wiki/File:Xerox_ColorQube_8570_-_Main_controller_-_Xilinx_Spartan_XC3S400A-0205.jpg)  **A Spartan FPGA from**[**Xilinx**](https://en.wikipedia.org/wiki/Xilinx)  FPGAs contain an array of [programmable](https://en.wikipedia.org/wiki/Programmable_logic_device) [logic blocks](https://en.wikipedia.org/wiki/Logic_block), and a hierarchy of "reconfigurable interconnects" that allow the blocks to be "wired together", like many logic gates that can be inter-wired in different configurations. [Logic blocks](https://en.wikipedia.org/wiki/Logic_block) can be configured to perform complex [combinational functions](https://en.wikipedia.org/wiki/Combinational_logic), or merely simple [logic gates](https://en.wikipedia.org/wiki/Logic_gate) like [AND](https://en.wikipedia.org/wiki/AND_gate) and [XOR](https://en.wikipedia.org/wiki/XOR_gate). In most FPGAs, logic blocks also include [memory elements](https://en.wikipedia.org/wiki/Memory_cell_(computing)), which may be simple [flip-flops](https://en.wikipedia.org/wiki/Flip-flop_(electronics)) or more complete blocks of memory. Many FPGAs can be reprogrammed to implement different [logic functions](https://en.wikipedia.org/wiki/Boolean_function), allowing flexible [reconfigurable computing](https://en.wikipedia.org/wiki/Reconfigurable_computing) as performed in [computer software](https://en.wikipedia.org/wiki/Software). FPGAs have a remarkable role in the embedded system development due to capability to   * start system software (SW) development simultaneously with hardware (HW) * enable system performance simulations at very early phase of the development * allow various system partitioning (SW and HW) trials and iterations before final freezing of the system architecture   The FPGA industry sprouted from [programmable read-only memory](https://en.wikipedia.org/wiki/Programmable_read-only_memory) (PROM) and [programmable logic devices](https://en.wikipedia.org/wiki/Programmable_logic_devices) (PLDs). PROMs and PLDs both had the option of being programmed in batches in a factory or in the field (field-programmable). However, programmable logic was hard-wired between logic gates.  [Altera](https://en.wikipedia.org/wiki/Altera) was founded in 1983 and delivered the industry's first reprogrammable logic device in 1984 – the EP300 – which featured a quartz window in the package that allowed users to shine an ultra-violet lamp on the die to erase the [EPROM](https://en.wikipedia.org/wiki/EPROM) cells that held the device configuration. In December 2015, [Intel](https://en.wikipedia.org/wiki/Intel) acquired Altera.  [Xilinx](https://en.wikipedia.org/wiki/Xilinx) co-founders [Ross Freeman](https://en.wikipedia.org/wiki/Ross_Freeman) and [Bernard Vonderschmitt](https://en.wikipedia.org/wiki/Bernard_Vonderschmitt) invented the first commercially viable field-programmable [gate array](https://en.wikipedia.org/wiki/Gate_array) in 1985 – the XC2064. The XC2064 had programmable gates and programmable interconnects between gates, the beginnings of a new technology and market. The XC2064 had 64 configurable logic blocks (CLBs), with two three-input [lookup tables](https://en.wikipedia.org/wiki/Lookup_table) (LUTs). More than 20 years later, Freeman was entered into the [National Inventors Hall of Fame](https://en.wikipedia.org/wiki/National_Inventors_Hall_of_Fame) for his invention.  In 1987, the [Naval Surface Warfare Center](https://en.wikipedia.org/wiki/Naval_Surface_Warfare_Center) funded an experiment proposed by Steve Casselman to develop a computer that would implement 600,000 reprogrammable gates. Casselman was successful and a patent related to the system was issued in 1992.  Altera and Xilinx continued unchallenged and quickly grew from 1985 to the mid-1990s, when competitors sprouted up, eroding significant market share. By 1993, Actel (now [Microsemi](https://en.wikipedia.org/wiki/Microsemi" \o "Microsemi)) was serving about 18 percent of the market. By 2013, Altera (31 percent), Actel (10 percent) and Xilinx (36 percent) together represented approximately 77 percent of the FPGA market.  The 1990s were a period of rapid growth for FPGAs, both in circuit sophistication and the volume of production. In the early 1990s, FPGAs were primarily used in [telecommunications](https://en.wikipedia.org/wiki/Telecommunication) and [networking](https://en.wikipedia.org/wiki/Computer_network). By the end of the decade, FPGAs found their way into consumer, automotive, and industrial applications.  Companies like Microsoft have started to use FPGAs to accelerate high-performance, computationally intensive systems (like the [data centers](https://en.wikipedia.org/wiki/Data_center) that operate their [Bing search engine](https://en.wikipedia.org/wiki/Bing_(search_engine))), due to the [performance per watt](https://en.wikipedia.org/wiki/Performance_per_watt) advantage FPGAs deliver. Microsoft began using FPGAs to [accelerate](https://en.wikipedia.org/wiki/Hardware_acceleration) Bing in 2014, and in 2018 began deploying FPGAs across other data center workloads for their [Azure](https://en.wikipedia.org/wiki/Microsoft_Azure) [cloud computing](https://en.wikipedia.org/wiki/Cloud_computing) platform. Complex Programmable Logic Devices (CPLD) The primary differences between [complex programmable logic devices](https://en.wikipedia.org/wiki/Complex_programmable_logic_device) (CPLDs) and FPGAs are [architectural](https://en.wikipedia.org/wiki/Computer_architecture). A CPLD has a comparatively restrictive structure consisting of one or more programmable [sum-of-products](https://en.wikipedia.org/wiki/Canonical_normal_form) logic arrays feeding a relatively small number of clocked [registers](https://en.wikipedia.org/wiki/Register_(computing)). As a result, CPLDs are less flexible, but have the advantage of more predictable [timing delays](https://en.wikipedia.org/wiki/Latency_(engineering)) and a higher logic-to-interconnect ratio. FPGA architectures, on the other hand, are dominated by [interconnect](https://en.wikipedia.org/wiki/Communications_subsystem). This makes them far more flexible (in terms of the range of designs that are practical for implementation on them) but also far more complex to design for, or at least requiring more complex [electronic design automation](https://en.wikipedia.org/wiki/Electronic_design_automation) (EDA) software.  In practice, the distinction between FPGAs and CPLDs is often one of size as FPGAs are usually much larger in terms of resources than CPLDs. Typically only FPGAs contain more complex [embedded functions](https://en.wikipedia.org/wiki/Functional_unit) such as [adders](https://en.wikipedia.org/wiki/Adder_(electronics)), [multipliers](https://en.wikipedia.org/wiki/Binary_multiplier), [memory](https://en.wikipedia.org/wiki/Computer_memory), and [serializer/deserializers](https://en.wikipedia.org/wiki/SerDes" \o "SerDes). Another common distinction is that CPLDs contain embedded [flash memory](https://en.wikipedia.org/wiki/Flash_memory) to store their configuration while FPGAs usually require external [non-volatile memory](https://en.wikipedia.org/wiki/Non-volatile_memory) (but not always).  When a design requires simple instant-on [(logic is already configured at power-up)](https://en.wikipedia.org/wiki/Glue_logic) CPLDs are generally preferred. For most other applications FPGAs are generally preferred. Sometimes both CPLDs and FPGAs are used in a single system design. In those designs, CPLDs generally perform glue logic functions, and are responsible for “[booting](https://en.wikipedia.org/wiki/Booting)” the FPGA as well as controlling [reset](https://en.wikipedia.org/wiki/Reset_(computing)) and boot sequence of the complete circuit board. Therefore, depending on the application it may be judicious to use both FPGAs and CPLDs in a single design.[[32]](https://en.wikipedia.org/wiki/Field-programmable_gate_array#cite_note-32) Security considerations FPGAs have both advantages and disadvantages as compared to ASICs or secure microprocessors, concerning [hardware security](https://en.wikipedia.org/wiki/Hardware_security). FPGAs' flexibility makes malicious modifications during [fabrication](https://en.wikipedia.org/wiki/Semiconductor_device_fabrication) a lower risk.[[33]](https://en.wikipedia.org/wiki/Field-programmable_gate_array#cite_note-paper-33) Previously, for many FPGAs, the design [bitstream](https://en.wikipedia.org/wiki/Bitstream" \o "Bitstream) was exposed while the FPGA loads it from external memory (typically on every power-on). All major FPGA vendors now offer a spectrum of security solutions to designers such as bitstream [encryption](https://en.wikipedia.org/wiki/Encryption) and [authentication](https://en.wikipedia.org/wiki/Authentication). For example, [Altera](https://en.wikipedia.org/wiki/Altera) and [Xilinx](https://en.wikipedia.org/wiki/Xilinx) offer [AES](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard) encryption (up to 256-bit) for bitstreams stored in an external flash memory.  FPGAs that store their configuration internally in nonvolatile flash memory, such as [Microsemi](https://en.wikipedia.org/wiki/Microsemi" \o "Microsemi)'s ProAsic 3 or [Lattice](https://en.wikipedia.org/wiki/Lattice_Semiconductor)'s XP2 programmable devices, do not expose the bitstream and do not need [encryption](https://en.wikipedia.org/wiki/Encryption). In addition, flash memory for a [lookup table](https://en.wikipedia.org/wiki/Lookup_table) provides [single event upset](https://en.wikipedia.org/wiki/Single_event_upset) protection for space applications.[[clarification needed](https://en.wikipedia.org/wiki/Wikipedia:Please_clarify)] Customers wanting a higher guarantee of tamper resistance can use write-once, [antifuse](https://en.wikipedia.org/wiki/Antifuse" \o "Antifuse) FPGAs from vendors such as [Microsemi](https://en.wikipedia.org/wiki/Microsemi" \o "Microsemi).  With its Stratix 10 FPGAs and SoCs, [Altera](https://en.wikipedia.org/wiki/Altera) introduced a Secure Device Manager and physically uncloneable functions to provide high levels of protection against physical attacks. Applications An FPGA can be used to solve any problem which is [computable](https://en.wikipedia.org/wiki/Computability). This is trivially proven by the fact that FPGAs can be used to implement a [soft microprocessor](https://en.wikipedia.org/wiki/Soft_microprocessor), such as the Xilinx [MicroBlaze](https://en.wikipedia.org/wiki/MicroBlaze" \o "MicroBlaze) or Altera [Nios II](https://en.wikipedia.org/wiki/Nios_II" \o "Nios II). Their advantage lies in that they are significantly faster for some applications because of their [parallel nature](https://en.wikipedia.org/wiki/Parallel_computing) and [optimality](https://en.wikipedia.org/wiki/Logic_optimization) in terms of the number of gates used for certain processes.  FPGAs originally began as competitors to [CPLDs](https://en.wikipedia.org/wiki/Complex_programmable_logic_device) to implement [glue logic](https://en.wikipedia.org/wiki/Glue_logic) for [printed circuit boards](https://en.wikipedia.org/wiki/Printed_circuit_board). As their size, capabilities, and speed increased, FPGAs took over additional functions to the point where some are now marketed as full [systems on chips](https://en.wikipedia.org/wiki/System_on_a_chip) (SoCs). Particularly with the introduction of dedicated [multipliers](https://en.wikipedia.org/wiki/Binary_multiplier) into FPGA architectures in the late 1990s, applications which had traditionally been the sole reserve of [digital signal processor hardware](https://en.wikipedia.org/wiki/Digital_signal_processor) (DSPs) began to incorporate FPGAs instead.  module NAND\_2\_gate\_level(output Y, input A, B);     wire Yd;     and(Yd, A, B);     not(Y, Yd);  endmodule |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **01-June-2020** | **Name:** | **Raziya Banu** | |
| **Course:** | **Udemy** | **USN:** | **4AL16EC058** | |
| **Topic:** | **Conditional statement in Python** | **Semester & Section:** | **8th sem & ‘B’ section** | |
| **AFTERNOON SESSION DETAILS** | | | |
| **Image of session** | | | |
| **Python Sets:**Conditional StatementsDecision MakingGender Decision **Conditions –**  usually in the form of if statements - are one of the key features of a programming language, and Python is no exception. You will hardly find a programming language without an if statement.1 There is hardly a way to program without branches in the code flow, at least if the code needs to solve a useful problem.  A decision must be made when the script or program comes to a point where there is a selection of actions, i.e. different calculations from which to choose.  The decision, in most cases, depends on the value of variables or arithmetic expressions. These expressions are evaluated using the Boolean True or False values. The instructions for decision making are called conditional statements. In a programming language, therefore, the parts of the conditional statements are executed under the given conditions. In many cases there are two code parts: One which will be executed, if the condition is True, and another one, if it is False. In other words, a branch determines which of two (or even more) program parts (alternatives) will be executed depending on one (or more) conditions. Conditional statements and branches belong to the control structures of programming languages, because with their help a program can react to different states that result from inputs and calculations.  **Conditional Statements in Real Life**  The principle of the indentation style is also known in natural languages, as we can deduce from the following text:  If it rains tomorrow, I'll clean up the basement.  After that, I will paint the walls. If there is  any time left, I will file my tax return.  Of course, as we all know, there will be no time left to file the tax return. Jokes aside: in the previous text, as you see, we have a sequence of actions that must be performed in chronological order. If you take a closer look at the text, you will find some ambiguity in it: is 'the painting of the walls' also related to the event of the rain? Will the tax declaration be done with or without rain? What will this person do, if it does not rain? Therefore, we extend the text with additional alternatives:  If it rains tomorrow, I'll clean up the basement.  After that I will paint the walls. If there is  any time left, I will file my tax return. Otherwise,  i.e. if it will not rain, I will go swimming.  In the evening, I will go to the cinema with my wife!  It is still nor clear. Can his wife hope to be invited to the cinema? Does she have to pray or hope for rain? To make the text clear, we can phrase it to be closer to the programming code and the Python coding style, and hopefully there will be a happy ending for his wife:  If it rains tomorrow, I will do the following:  - Clean up the basement  - Paint the walls  - If there is any time left, I will make my  tax declaration  Otherwise I will do the following:  - Go swimming  Going to the movies in the evening with my wife  We can also graphically depict this. Such a workflow is often referred to in the programming environment as a so-called flowchart or program schedule (PAP):  Flowchart Example  To encode conditional statements in Python, we need to know how to combine statements into a block. So this seems to be the ideal moment to introduce the Python Block Principle in combination with the conditional statements.  **Combining Statements into Blocks**  In programming, blocks are used to treat sets of statements as if they were a single statement. A block consists of one or more statements. A program can be considered a block consisting of statements and other nested blocks. In programming languages, there exist various approaches to syntactically describing blocks: ALGOL 60 and Pascal, for example, use "begin" and "end", while C and similar languages use curly braces "{" and "}". Bash has another design that uses 'do ... done' and 'if ... fi' or 'case ... esac' constructs. All these languages have in common is that the indentation style is not binding. The problems that may arise from this will be shown in the next subchapter. You can safely skip this, if you are only interested in Python. For inexperienced programmers, this chapter may seem a bit difficult.  **Excursion to C**  For all these approaches, there is a big drawback: The code may be fine for the interpreter or compiler of the language, but it can be written in a way that is poorly structured for humans. We want to illustrate this in the following "pseudo" C-like code snippet:  if (raining\_tomorrow) {  tidy\_up\_the\_cellar();  paint\_the\_walls();  if (time\_left)  do\_taxes();  } else  enjoy\_swimming();  go\_cinema();  We will add a couple of spaces in front of calling 'go\_cinema'. This puts this call at the same level as 'enjoy\_swimming':  if (raining\_tomorrow) {  tidy\_up\_the\_cellar();  paint\_the\_walls();  if (time\_left)  do\_taxes();  } else  enjoy\_swimming();  go\_cinema();  The execution of the program will not change: you will go to the cinema, whether it will be raining or not! However, the way the code is written falsely suggests that you will only go to the cinema if it does not rain. This example shows the possible ambiguity in the interpretation of C code by humans. In other words, you can inadvertently write the code, which leads to a procedure that does not "behave" as intended.  In this example lurks even another danger. What if the programmer had just forgotten to put the last statement out of the curly braces?  Should the code look like this:  if (raining\_tomorrow) {  tidy\_up\_the\_cellar();  paint\_the\_walls();  if (time\_left)  do\_taxes();  } else {  enjoy\_swimming();  go\_cinema();  }  In the following program you will only go to the cinema, if it is not a rainy day. That only makes sense if it's an open-air cinema.  The following code is the right code to go to the movies regardless of the weather:  if (raining\_tomorrow) {  tidy\_up\_the\_cellar();  paint\_the\_walls();  if (time\_left)  do\_taxes();  } else {  enjoy\_swimming();  }  go\_cinema();} | | | |