Ans.1 Moore's Law is the observation made in 1965 by Gordon Moore, co-founder of [Intel](https://www.webopedia.com/TERM/I/Intel.html), that the number of [transistors](https://www.webopedia.com/TERM/T/transistor.html) per square inch on [integrated circuits](https://www.webopedia.com/TERM/I/integrated_circuit_IC.html) had doubled every year since the integrated circuit was invented. Moore predicted that this trend would continue for the foreseeable future. In subsequent years, the pace slowed down a bit, but data density has doubled approximately every 18 months, and this is the current definition of Moore's Law, which Moore himself has blessed. Most experts, including Moore himself, expect Moore's Law to hold true until 2020-2025.

Ans.2 he Ychart in VLSI has been developed by Gajski and Kuhn in the year 1983 to categorise the behaviour of hardware designs on the basis of the three different domains.

Following is the representation of the Gajski-kuhn Y chart.

The three different domains are ***behavioural,structural and physical/geometry domain*** which are on radial axis.Each of the corresponding domains can be further divided into levels of abstraction using concentric rings and each of the domain falling within the circle forms a group and keep going on in a top down fashion towards the centre of the core.

Let's discuss each domain in a bit detail:-

**1] Physical/geometry domain:-**

At the physical domain it is first necessary to subdivide a large system into small ASIC sized pieces which is referred as ***physical partition***.After that the small blocks are arranged together in the form of ***clusters***.Then the next step is ***floor plan*** where it is necessary to find out the approximate location of each module or block in the chip.After that we can move on to ***module layout and cell layout*** where each unit of block to be placed on chip and the connections between the cells and the block is defined.

**2] Behavioural domain:-**

In this domain the behaviour of the entity can be modelled using procedural code which can represent the circuit at a higher level of abstraction.

In this domain the first hierarchy is ***system*** where the behaviour of the architecture is defined.Then a set of ***algorithm*** needs to be defined for processing the different parameters and a set of flowchart.The third step is ***RTL*** where the coding needs to be done and then the ***logic*** level where we are concerned with 0 and 1 level,after which the ***transfer function*** is defined.

Generally the people who are more interested how the system works entered into this field.

**3] Structural domain:-**

In this domain a particular block is connected across with set of signals or netlist and here in this case we are more interested in the structure.

Generally the people in this field are more interested to know how a system works. So all CPU design,verification related works come in this field as this domain is more inclined towards processors and memories.

So the above description summarises the concept behind Y chart.

Now let's consider the Y chart first without any kind of circle drawn on it.The point which I am trying to make is that when we draw the circle,each one of them cuts through different points across the three different domains forming a group and the process is repeated as we keep drawing the circle.

Mostly, it is used for the development of integrated circuits.According to this model, the development of hardware is perceived within three domains that are depicted as three axis and produce a Y. Along these axis, the abstraction levels that describe the degree of abstraction. The outer shells are generalisations , the inner ones refinements of the same subject.

The issue in hardware development is most often a top-down design problem. This is perceived by the three domains of behaviour , structure, and the layout that goes top-down to more detailed abstraction levels. The designer can select one of the perspectives and then switch from one view to another. Generally, the design process is not following a specific sequence in this diagram.

* On the **system level**, basic properties of an electronic system are determined. For the behavioural description, block diagrams are used by making abstractions of signals and their time response. Blocks used in the structure domain are CPUs , memory chip etc.
* The **algorithmic level** is defined by the definition of concurrent algorithms (signals, loops, variables, assignments). In the structural domain, blocks like ALUs are in use.
* The **register-transfer level** (RTL) is a more detailed abstraction level on which the behaviour between communicating registers and logic units is described. Here, data structures and data flows are defined. In the geometric view, the design step of the floorplan is located.
* The **logical level** is described in the behaviour perspective by boolean equations. In the structural view, this is displayed with gates and flip-flops. In the geometric domain, the logical level is described by standard cells.
* The behaviour of the **circuit level** is described by mathematics using different equations or logical equations. This corresponds to transistors and capacitors up to crystal lattices.
* also known as Y-chart represents 3 views of a hardware design model. I am not sure how much this is applied practically.
* A good practice in any system level HW+SW design is to separate/isolate the “application/usage model” from “architecture” and “implementation” details and find the right mapping between them. This is what the chart summarizes.
* In terms of abstraction, the outer ring in the chart is the highest level of abstraction and it decreases as you move to the center of the chart. It is captured as “System level” in the outer ring and as you move down - “algorithmic” to “RTL” to “logic” to “circuit” level.
* Normally you start your system design with a behavioral view with high level abstraction in terms of System Specification. You would then break it down into lower levels of abstractions like Algorithms and low level functions, boolean equations etc that match your design goals.
* The second part is to map this behavioral view of application/usage model into an architecture - in terms of CPUs, memories, co-processors and further low level abstractions (FSM , ALU , flip flop etc) which is the second axis across Structural domain.
* The third view is mapping this to a physical implementation in terms of systems, boards, chips, floor plan, layout etc

Ans 1. VLSI Design Styles

VLSI design styles are primarily classified as full custom and semi custom design styles.In the former case, the functional and physical designs are fully customized, requiring anextensive effort of a design team to optimize each detailed feature of the circuit. In thiscase, the design effort and cost are high, often compensated by the achievement of highquality circuits. It is obvious that the cost has to be amortized over large volume production (as in the case of processor design) or borne in full (as in case of specialized processor design).In case of semi-custom VLSI design, some part of the design is already being made and put in the library. The designer is required to define the interconnections between thosemodules. The reduction of the possible number of implementation choices makes it easier to develop CAD tools for design and optimization and also reducing the design time.Semi-custom design may be further classified into two subclasses: cell based design andarray based design. Cell based design make use of library cells. Library cells are designedonce and stored. Cell based designs are further classified as cell library (standard cell) based design and macro cell based design.In standard cell based designs, cells are designed once but updates are required as progress in semiconductor technology allows for smaller geometries. Every cell needs to be parameterized in terms of area and delay, over ranges of temperatures and operatingvoltages.Macro cell based design consists of combining building blocks that can be synthesized bycomputer programs called cell generators. Originally the macro cell generators targetedthe automated synthesis of memory arrays and programmable logic arrays. The recentmacro cell generators start with the functional description of the circuits and are able tosynthesize the layout of those circuits that closely match their optimized full customcounterpart.Array based designs are further classified as pre-diffused and pre-wired, also called mask  programmable gate arrays and field programmable gate arrays. MPGA consists of  batches of wafers with array of sites being manufactured. The fabrication process entails programming the sites by contacting them to wires, i.e. by contacting them to routingwires.FPGA, in contrast, is programmed on field, i.e. by applying electrical pulses. Theyconsist of arrays of programmable modules capable of implementing logic functionsconnected by switched wires. Programming is achieved in two ways. Wires, already present in the form of segments can be connected by programming anti-fuses.Alternatively, memory elements inside the array can be programmed to store informationthat relates to the module configuration and interconnection.

**Ans1.what is a Programmable Logic Device?**

A PLD, or programmable logic device, is an electronic component that is used in order to build digital circuits that are reprogrammable. A programmable logic device does not have a defined function once manufactured, unlike a logic gate and has to be programmed before it can be used.

**Types of Programmable Logic Devices**

There are several different kinds of programmable logic devices at Future Electronics. We stock many of the most common types categorized by several parameters including Programmable Type, Number of I/O Lines, Supply Voltage, Memory Density, System Gates, Packaging Type and many other parameters specific to the type of programmable logic device. Our parametric filters will allow you to refine your search results according to the required specifications.

A field-programmable gate array (FPGA) is an integrated circuit ([IC](https://whatis.techtarget.com/definition/integrated-circuit-IC)) that can be programmed in the field after manufacture. FPGAs are similar in principle to, but have vastly wider potential application than, programmable read-only memory ([PROM](https://whatis.techtarget.com/definition/programmable-read-only-memory-PROM)) chips. FPGAs are used by engineers in the design of specialized ICs that can later be produced hard-wired in large quantities for distribution to computer manufacturers and end users. Ultimately, FPGAs might allow computer users to tailor microprocessors to meet their own individual needs.

**Advantages and Disadvantages of PLAs**

PLAs, like ROMs which are more general, have the following advantages over random-logic gate networks, where random-logic gate networks are those that are compactly laid out on an IC chip:

1. There is no neeed for the time-consuming logic design of random-logic gate networks and even more time-consuming layout.

2. Design checking is easy, and design change is also easy.

Layout is far simpler than that for random-logic gate networks, and thus is far less time-consuming.

4. When new IC fabrication technology is introduced, we can use previous design information with ease but without change, making adoption of the new technology quick and easy.

5. Only the connection mask needs to be custom-made.

6. Considering all these, PLA is a very inexpensive approach, greatly shortening desing time. PLAs have the following disadvantages compared with random-logic gate networks:

1. Random-logic gate networks have higher speed than PLAs or ROMs.

2. Random-logic gate networks occupy smaller chip areas than PLAs or ROMs, although the logic design and the layout of random-logic gate networks are far more tedious and time- consuming.

3. Also, with large production volumes, random-logic gate networks are cheaper than PLAs or ROMs.

PLAs have the following advantage and disadvantage, compared with ROMs:

• For storing the same functions or tasks, PLAs can be smaller than ROMs; generally, the size difference sharply increases as the number of input variables increases.

• The small size advantages of PLAs diminishes as the number of terms in a disjunctive form increases. Thus, PLAs cannot store complex functions, i.e., functions whose disjunctive forms consist of many product terms.

*Ans 2.Syntheis* is the translation process from a descritpion of a hardware device at higher abstraction level into an optimized implemenation on a lower level abstraction. This process may be done by human or a computer program. There is a surge of incentive to program automatic synthesis programs for VLSI designs because (1)VLSI complexity has increased tremendously recently;(2) VLSI technology has become mature and a work horse for many applications; (3) Demand for shorter and shorter design cycle (time to market); (4) Need to explore bigger design space across levels of abstraction. There are two general catagories of the synthesis process: (1) Behavior-to-Structure; (2)Structure-to-Physical Layout. We can also view the synthesis process in three diffrerent levels:(1) Behavior Synthesis; (2)Logic Synthesis; (3)Physical Synthesis.

**Logic Synthesis**

Logic synthesis involves both combinational and Sequential logic design. If a statbel diagram or table is given it goes through the state minimization process and then the state encoding process. After the encoding is completed truth tables or boolean expressions can be used to describe the relationships between input, current state and output and next state. The truthe table or/and boolean expression is optimized. The next step invloves the process of technology mapping which converts the optimized truth table or boolean expression into gates of a particular technology.

In [computer science](https://en.wikipedia.org/wiki/Computer_science), a **binary decision diagram** (**BDD**) or **branching program** is a [data structure](https://en.wikipedia.org/wiki/Data_structure) that is used to represent a [Boolean function](https://en.wikipedia.org/wiki/Boolean_function). On a more abstract level, BDDs can be considered as a [compressed](https://en.wikipedia.org/wiki/Data_compression) representation of [sets](https://en.wikipedia.org/wiki/Set_%28mathematics%29) or [relations](https://en.wikipedia.org/wiki/Relation_%28mathematics%29). Unlike other compressed representations, operations are performed directly on the compressed representation, i.e. without decompression. Other [data structures](https://en.wikipedia.org/wiki/Data_structures) used to represent a [Boolean function](https://en.wikipedia.org/wiki/Boolean_function) include [negation normal form](https://en.wikipedia.org/wiki/Negation_normal_form) (NNF), and [propositional directed acyclic graph](https://en.wikipedia.org/wiki/Propositional_directed_acyclic_graph) (PDAG).

A Boolean function can be represented as a [rooted](https://en.wikipedia.org/wiki/Rooted_graph), directed, acyclic [graph](https://en.wikipedia.org/wiki/Graph_theory), which consists of several decision nodes and terminal nodes. There are two types of terminal nodes called 0-terminal and 1-terminal. Each decision node N {\displaystyle N} is labeled by Boolean variable V N {\displaystyle V\_{N}} and has two [child nodes](https://en.wikipedia.org/wiki/Child_node) called low child and high child. The edge from node V N {\displaystyle V\_{N}} to a low (or high) child represents an assignment of V N {\displaystyle V\_{N}} to 0 (resp. 1). Such a **BDD** is called 'ordered' if different variables appear in the same order on all paths from the root. A BDD is said to be 'reduced' if the following two rules have been applied to its graph:

* Merge any [isomorphic](https://en.wikipedia.org/wiki/Graph_isomorphism) subgraphs.
* Eliminate any node whose two children are [isomorphic](https://en.wikipedia.org/wiki/Graph_isomorphism).

In popular usage, the term **BDD** almost always refers to **Reduced Ordered Binary Decision Diagram** (**ROBDD** in the literature, used when the ordering and reduction aspects need to be emphasized). The advantage of an ROBDD is that it is canonical (unique) for a particular function and variable order.[[1]](https://en.wikipedia.org/wiki/Binary_decision_diagram#cite_note-Bryant1986-1) This property makes it useful in functional equivalence checking and other operations like functional technology mapping.

A path from the root node to the 1-terminal represents a (possibly partial) variable assignment for which the represented Boolean function is true. As the path descends to a low (or high) child from a node, then that node's variable is assigned to 0 (resp. 1).

**Logical operations on BDDs**

Many logical operations on BDDs can be implemented by polynomial-time graph manipulation algorithms:[[16]](https://en.wikipedia.org/wiki/Binary_decision_diagram#cite_note-16):20

* [conjunction](https://en.wikipedia.org/wiki/Logical_conjunction)
* [disjunction](https://en.wikipedia.org/wiki/Logical_disjunction)
* [negation](https://en.wikipedia.org/wiki/Negation)
* existential abstraction[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia%3ACitation_needed)]
* universal abstraction[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia%3ACitation_needed)]

However, repeating these operations several times, for example forming the conjunction or disjunction of a set of BDDs, may in the worst case result in an exponentially big BDD. This is because any of the preceding operations for two BDDs may result in a BDD with a size proportional to the product of the BDDs' sizes, and consequently for several BDDs the size may be exponential. Also, since constructing the BDD of a Boolean function solves the NP-complete [Boolean satisfiability problem](https://en.wikipedia.org/wiki/Boolean_satisfiability_problem) and the co-NP-complete [tautology problem](https://en.wikipedia.org/wiki/Tautology_%28logic%29), constructing the BDD can take exponential time in the size of the Boolean formula even when the resulting BDD is small.