1. Embedded Systems definition, specification, and languages.

An embedded system is nearly any computing system other than a desktop computer.

An embedded system is a dedicated system which performs the desired function upon power up, repeatedly.

Embedded systems are found in a variety of common electronic devices such as consumer electronics ex. Cell phones, pagers, digital cameras, VCD players, portable Video games, calculators, etc.

Structure-oriented models describe the system's physical modules and the interconnections between them.

The major areas of the design process are

I. Ensuring a sound software and hardware specification.
II. Formulating the architecture for the system to be designed.
III. Partitioning the h/w and s/w.
IV. Providing an iterative approach to the design of h/w and s/w.

The important steps in developing an embedded system are

I. Requirement definition.
II. System specification.
III. Functional design
IV. Architectural design
V. Prototyping.

The system design specification

The System Design Specification (SDS) is a complete document that contains all of the information needed to develop the system.

Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements.

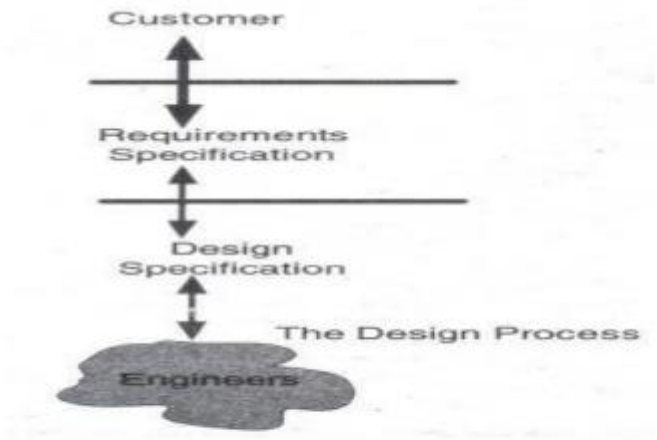
Systems design could be seen as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering.

System design specification serves as a bridges between the customers and designers.

It must specify the system‘s public interface from inside the system.

Five areas should be considered are:

I. Geographical constraints.
II. Characterization of and constraints on interface signals.
III. User interface requirements Temporal constraints.
IV. Electrical infrastructure consideration
V. Safety and reliability



The Customer, the requirement, the design and the engineer

Best Embedded Systems Programming Languages

Embedded Systems Programming is an exclusive industry and only a few programming languages are allowed entry because there are specific requirements such as low usage of resources as well as low-level system access.

1. C

- The loose data typing policy of C makes it quite suitable.
- It is simple to port the embedded programs from one device to the next as compared to other languages.
- The widespread community in C provides vast support for Embedded Systems Programming.

2. C++
    - C++ is less popular than C in regards to embedded systems but the addition of object-oriented programming.
    - It also has low-level access to memory like C that makes it quite suitable.
- C++ is more secure than C because of its use of string literals, enumeration constants, templates etc.
- The object oriented nature of C++ is also quite useful for complex embedded systems programming.

3. Java
- Java can be used to write extensible, portable, and downloadable embedded systems applications.
- There are many DevOps tools and libraries in Java that make it suitable for Embedded Systems Programming.
- The Java Virtual Machine ensures that embedded systems programmed in Java are portable and can be used for different IoT platforms.

4. Python
- Python is a popular language and known for its writability, concise, readable coding style, and error deduction.
- Python is much handier in the case of complicated embedded systems such as those using neural networks.
- Real time embedded systems use Python quite often. *MicroPython* is a good example of a lean and efficient implementation for this.

5. **Rust**
- Rust allows memory management using both dynamic and static methods using various tools.
- Rust can be used to program a wide range of embedded systems from small micro controllers to large multifaceted systems.
- There is a large community support in Rust for embedded systems programming.

6. **Ada**
- Ada is useful for embedded systems programming because of strong typing, run-time checking, parallel processing, exception handling, generics etc.
- Ada packages can be compiled separately as it was created for the development of large software systems.

7. Lua

- The base language in Lua is quite light as it has various meta-features that can be extended as required.
- Lua can implement object oriented programming using first-class functions and tables.
- Lua is cross-platform and it supports a C API that can be embedded into applications.

8. B#

- B# is well suited for small scale embedded systems programming because of its tiny core and small memory constraints.
- B# is type safe and does not allow any pointer manipulation which is an asset in embedded systems programming.
- The code written in B# is directly mapped to a tight instruction set. This reduces the runtime in low resource embedded devices.

9. Verilog

- Verilog can be used to design custom hardware that is required by the embedded system.
- Verilog has a module hierarchy wherein the modules can communicate with each other using the input, output, and bidirectional ports.
- The modules contain concurrent and sequential statement blocks. These blocks are executed concurrently and so Verilog is a dataflow language.

10. Embedded C++

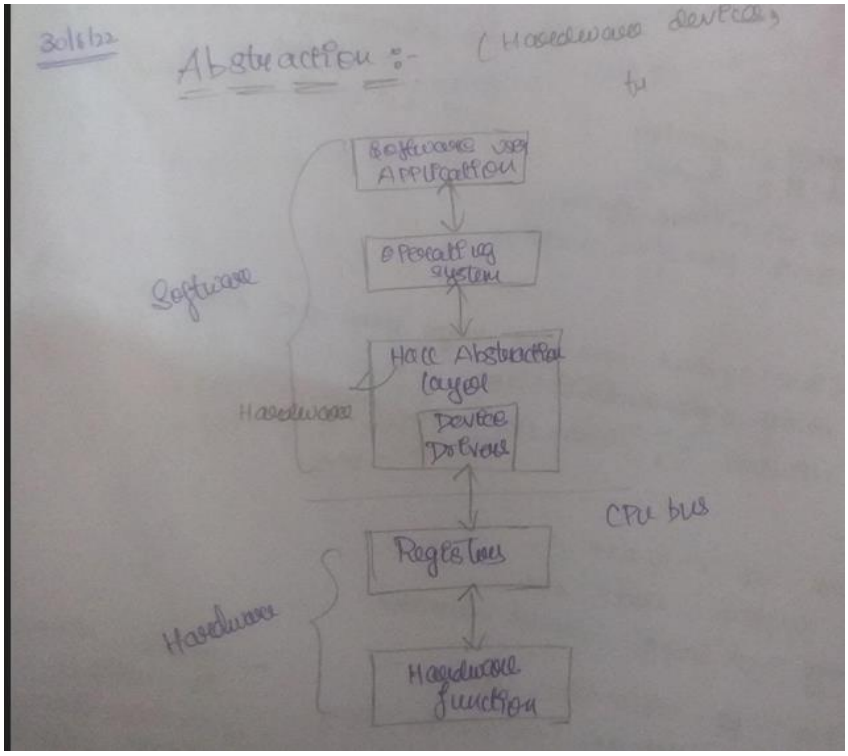
- Embedded C++ aims to minimize the code size and maximize the execution efficiency in regards to embedded systems programming.
- A standard C++ compiler can be used to compile the embedded systems programming done in Embedded C++.
- Some of the specific compilers for Embedded C++ are provided by Freescale Semiconductor, Green Hills Software etc.

2. Embedded system models at different abstraction levels.

Levels of abstraction

ABSTRATION:

- A level of abstraction is usually understood in terms of the language of design. Often, the language defines the level, but that doesn't have to be the case.
- A level of abstraction is determined by the objects that can be manipulated and the operations that can be performed on them.
- In programming terms, the objects are data types and the operations are the operators that can be used in expressions and control constructs.



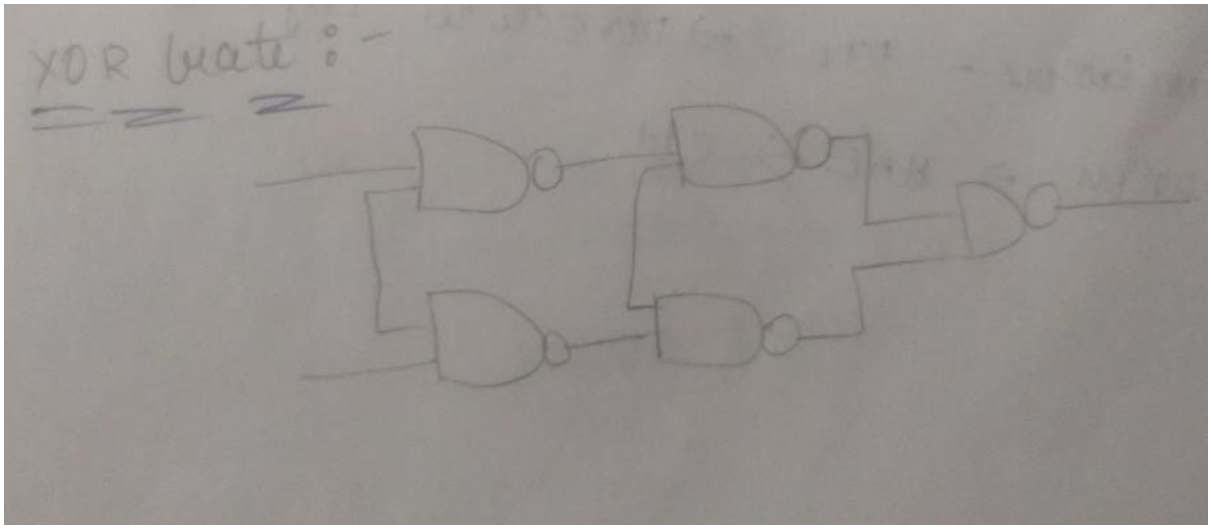
Interpretation

Interpretation is a technique for dynamically realizing the semantics of a language.

- A good example of a level of abstraction being implemented by interpretation is a micro-coded processor.
- The instructions visible to the programmer are implemented by a lower level processor which interprets the programmer-visible instructions.
- You can also make the argument that the use of a low-level VLIW processor to implement an algorithm is an example of interpretation. There aren't many other examples of the use of interpretation in hardware design.

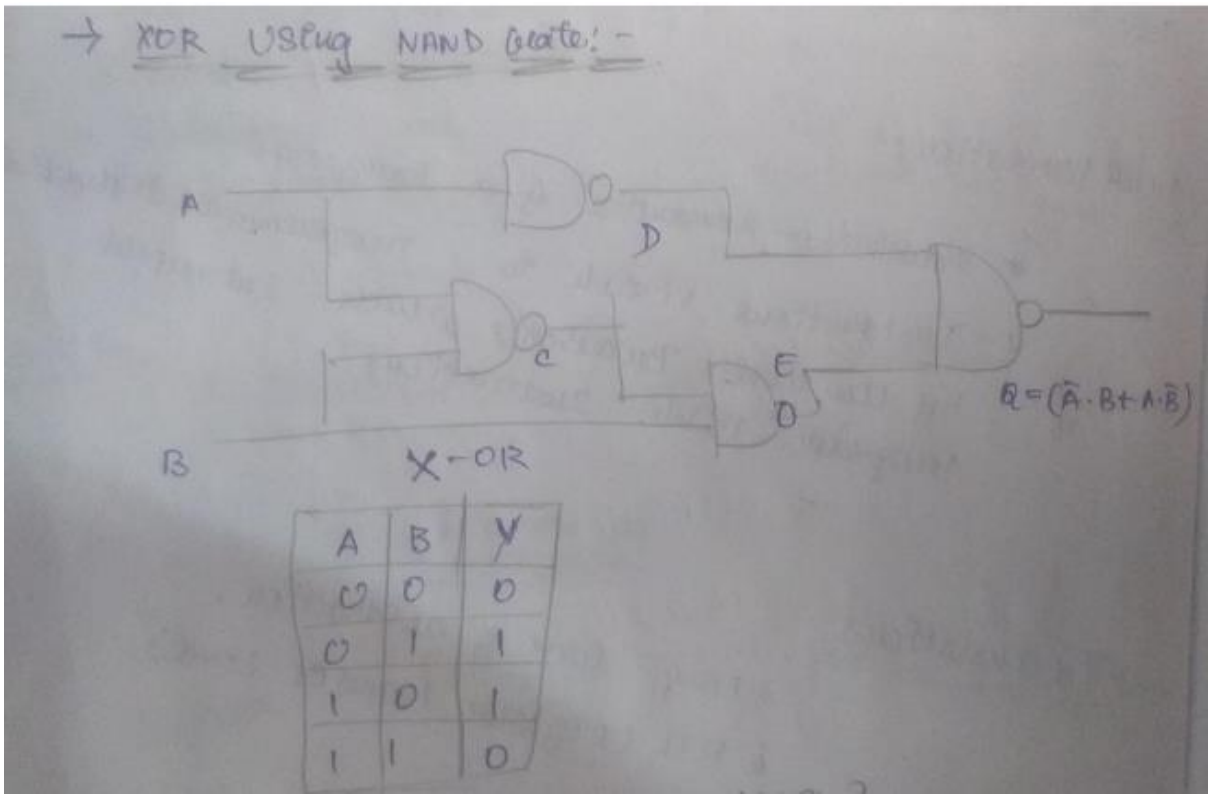
- **Translation**

  Translation is the most easily recognized form of abstraction implementation. Synopsys' Design Compiler translates RTL Verilog into a netlist, a process called logic synthesis.
- There has never been a widely accepted definition of the register-transfer level. The traditional definition is that RTL is whatever Design Compiler accepts.
- A better definition is that RTL is a representation where the activity in each clock cycle is explicitly defined.
- Behavioral synthesis was an easily identified analog of logic synthesis. It translated language at a behavioral level to RTL (or directly to gates).



- **Extension**

  Extension is not as widely recognized in hardware design as translation and interpretation, but is potentially the most useful.
- Extension is accomplished by defining new data types and operations in the same language that is used in the next lower level.
- Object oriented languages make creating new levels of abstraction in a single language particularly easy, and in fact, that is the primary virtue of C++.
- SystemC is nothing more than a hardware level of abstraction implemented by extension in C++.
- More important is that using C++, additional data types and operations can be defined to create higher levels than the base SystemC level.



3. REAL-TIME OPERATING SYSTEM (RTOS)

- RTOS is a multitasking operating system intended for real time applications.
- In the OS, there is a module called the scheduler, which schedules different tasks and determines when a process will execute on the processor.Multi-tasking is achieved in this way.
- The scheduler in RTOS is designed to provide a predictable execution pattern. In an embedded system, a certain event must be carried across strictly defined time.
- To meet real time requirements, the behavior of scheduler must be predictable.
- This type of OS which have a scheduler with predictable execution pattern is called real time OS (RTOS). The features of an RTOS are:

1. Context switching latency should be short.
2. Interrupt latency should be short.
3. Interrupt dispatch latency should be short.
4. Reliable and time bound inter process mechanism.
5. Support kernel pre-emption.

Types of real-time systems based on timing constraints:

1. **Hard real-time system:** This type of system can never miss its deadline. Missing the deadline may have disastrous consequences. The usefulness of results produced by a hard real-time system decreases abruptly and may become negative if tardiness increases. Tardiness means how late a real-time system completes its task with respect to its deadline. Example: Flight controller system.
2. **Soft real-time system:** This type of system can miss its deadline occasionally with some acceptably low probability. Missing the deadline have no disastrous consequences. The usefulness of results produced by a soft real-time system decreases gradually with an increase in tardiness. Example: Telephone switches.

Real-time systems are used in:

- Airlines reservation system.
- Air traffic control system.
- Systems that provide immediate updating.
- Used in any system that provides up to date and minute information on stock prices.
- Defense application systems like RADAR.
- Networked Multimedia Systems
- Command Control Systems
- Internet Telephony
- Anti-lock Brake Systems
- Heart Pacemaker