ASIC, FPGA and RISC-V

1.1 IC Transistors Density

Based on the number of components used (typically based on the number of transistors used), the ICs are categorized as follows:

Small-scale Integration: consists of only a few transistors (tens of transistors on a chip), these ICs played a critical role in early aerospace projects.

Medium-scale Integration: consists of some hundreds of transistors on the IC chip developed in the 1960s and achieved better economy and advantages compared to the SSI ICs.

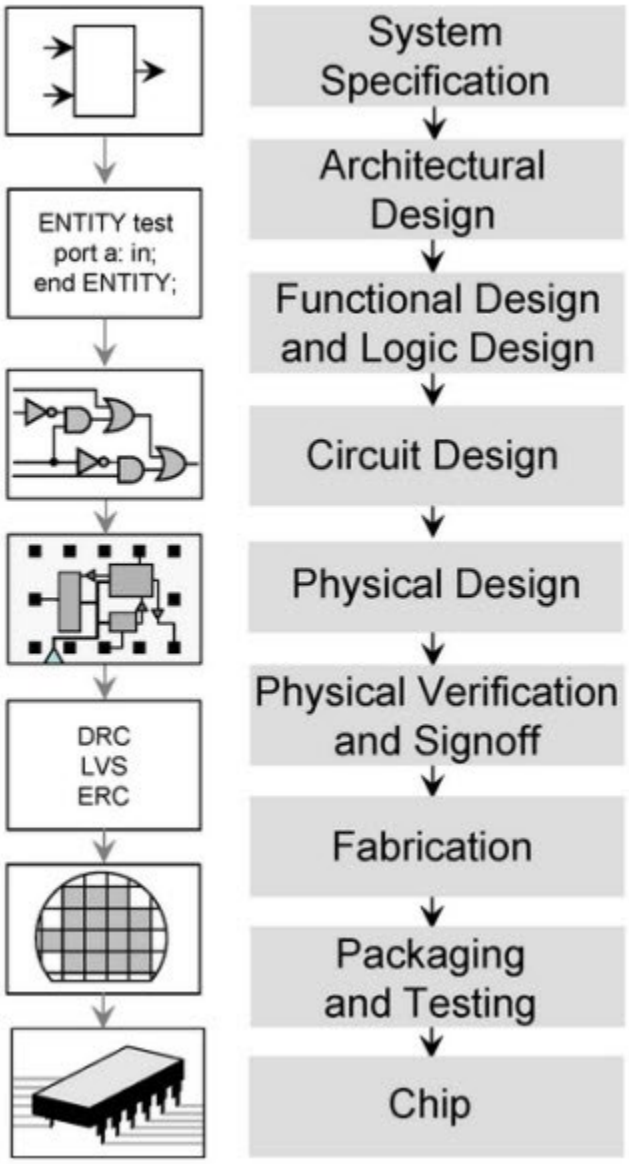
Large-scale Integration: consists of thousands of transistors on the chip with almost the same economy as medium-scale integration ICs. The first microprocessor, calculator chips, and RAMs of 1Kbit developed in the 1970s had below four thousand transistors.

Very Large-scale Integration: consists of transistors from hundreds to several billion in number (Development period: from the 1980s to 2009).

Ultra large-scale Integration: consists of transistors in excess of more than one million, and later wafer-scale integration (WSI), system on a chip (SoC), and three-dimensional integrated circuit (3D-IC) were developed.

Density of Integration / Complexity	Gates per IC
SSI: Small-Scale Integration • Logic Gates (AND, OR, NAND, NOR)	<10
MSI: Medium-Scale Integration • Flip Flops • Adders / Counters • Multiplexers & De-multiplexers	10 – 100
LSI: Large-Scale Integration •Small Memory Chips •Programmable Logic Device	100 – 10,000
VLSI: Very Large-Scale Integration •Large Memory Chips •Complex Programmable Logic Device	10,000 – 100,000
ULSI: Ultra Large-Scale Integration •8 & 16 Bit Microprocessors	100,000 – 1,000,000
GSI: Giga-Scale Integration •Pentium IV Processor	>1,000,000

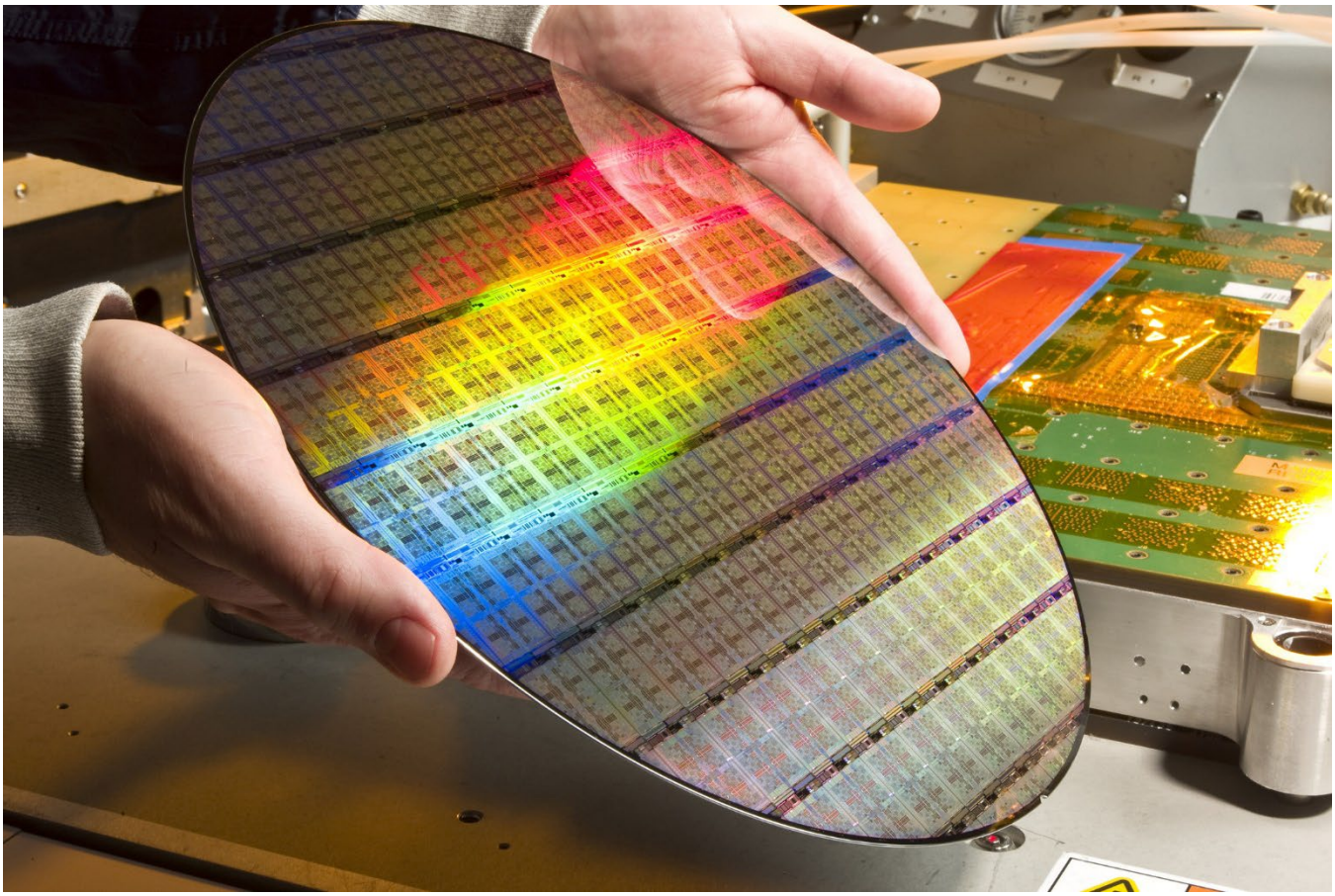
1.1.1 Integrated Circuit (IC) Design Flow Diagram



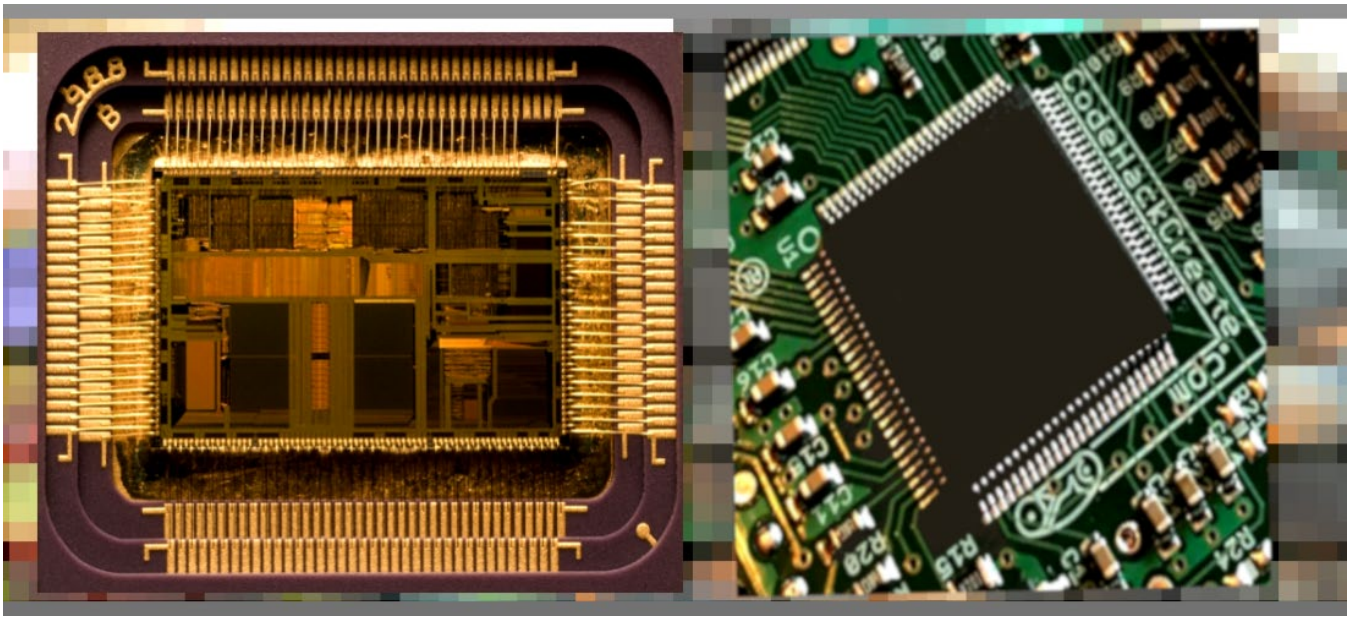
1.1.1 Silicon Wafer

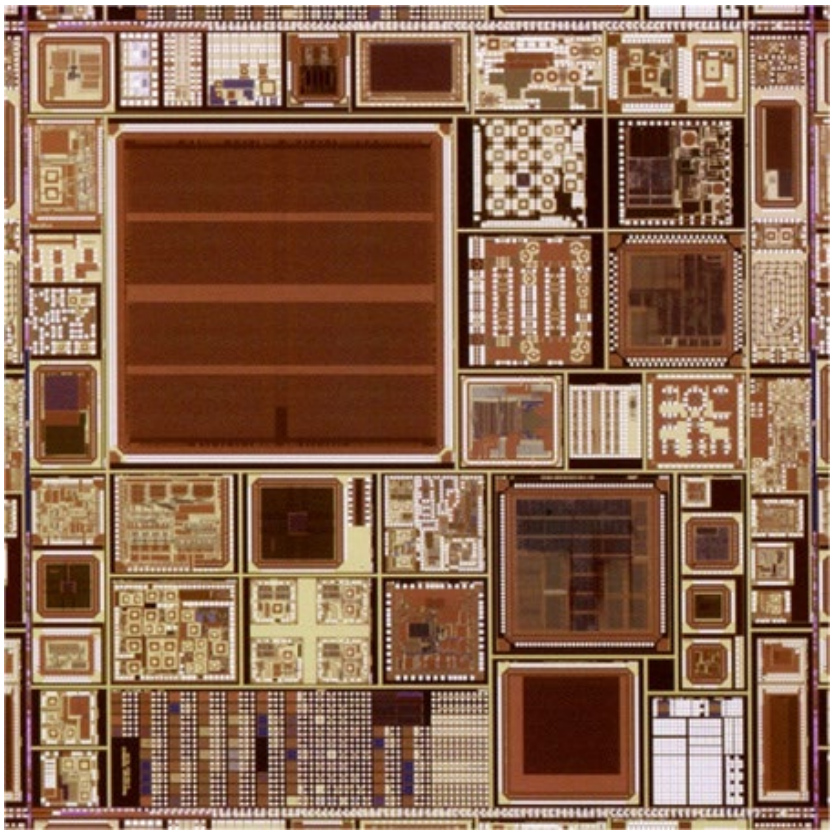
In electronics, a wafer (also called a slice or substrate) is a thin slice of semiconductor, such as a crystalline silicon (c-Si), used for the fabrication of integrated circuits. The wafer serves as the substrate for microelectronic devices built in and upon the wafer.

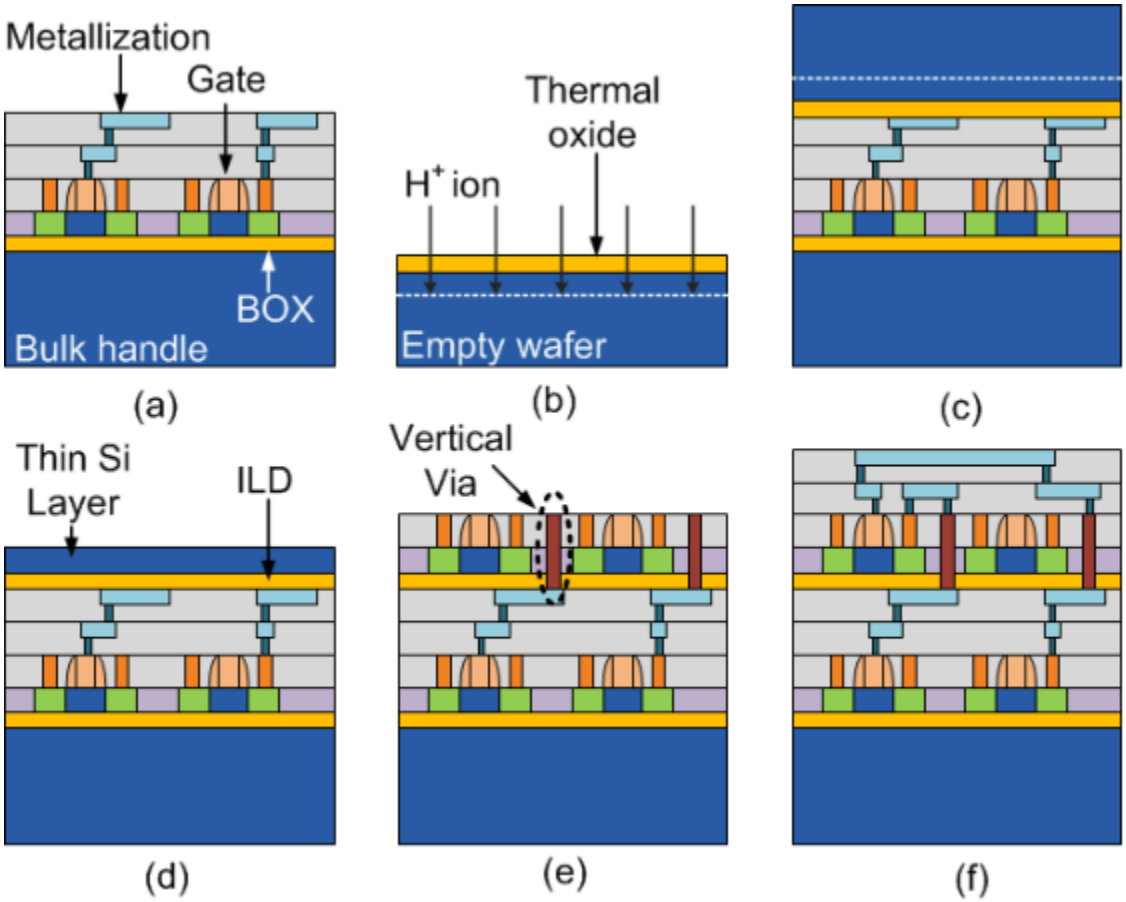
It undergoes many microfabrication processes, such as doping, ion implantation, etching, thin-film deposition of various materials, and photolithographic patterning. Finally, the individual microcircuits are separated by wafer dicing and packaged as an integrated circuit.



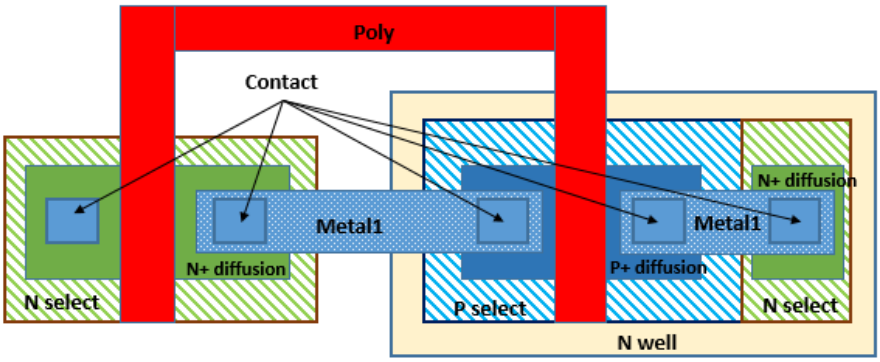
The following diagrams shows the internal design of an IC chip fabricated on the above shown silicon wafer:



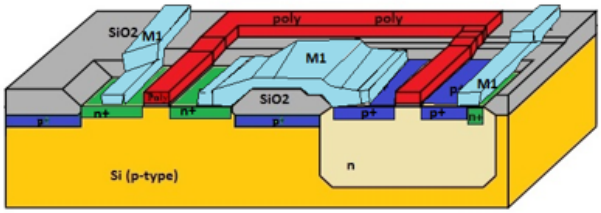



Metallization
Gate
BOX
Bulk handle
(a)
Thermal
oxide
H+ ion
Empty wafer
(b)
(c)
Thin Si
Layer
ILD
(d)
Vertical
Via
(e)
(f)

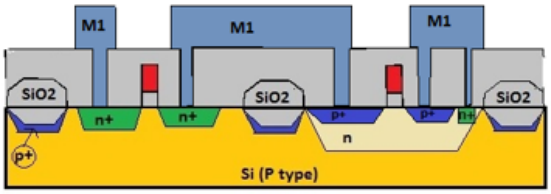
Top View

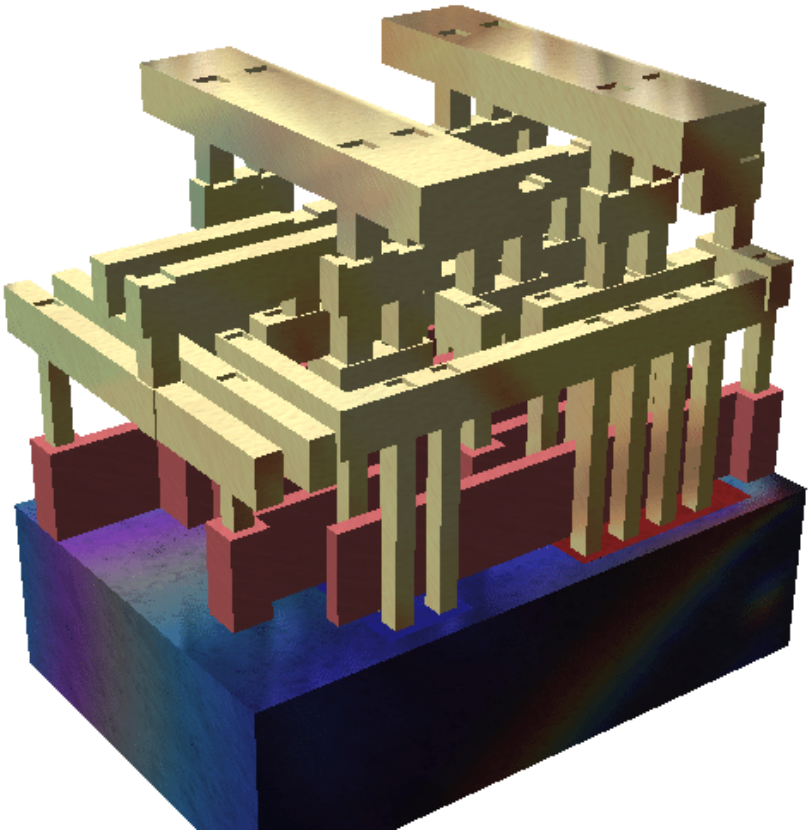


3D View

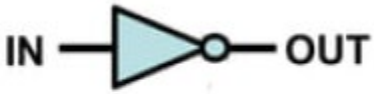


Side View

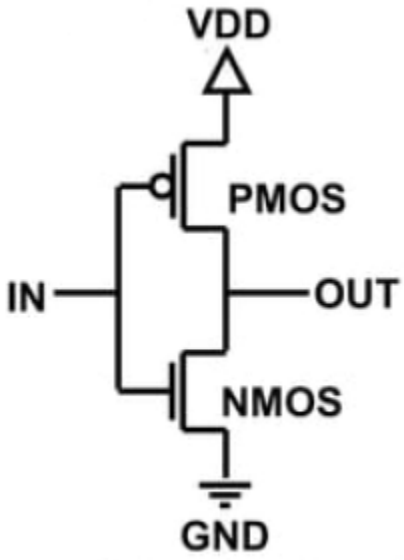




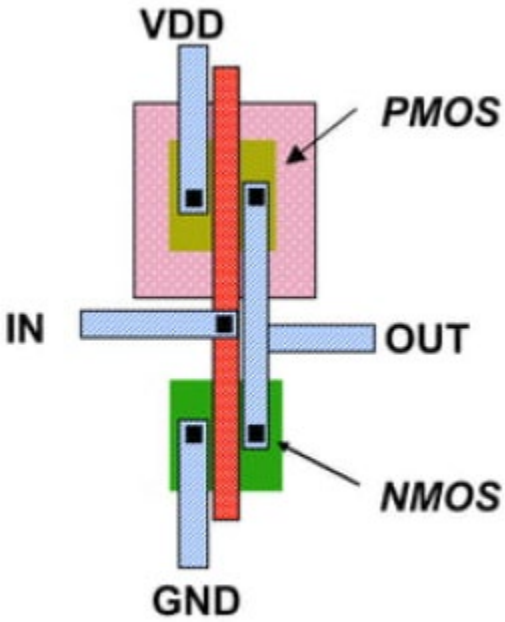
1.1.2 Example: NOT Gate Fabrication on the Silicon Wafer



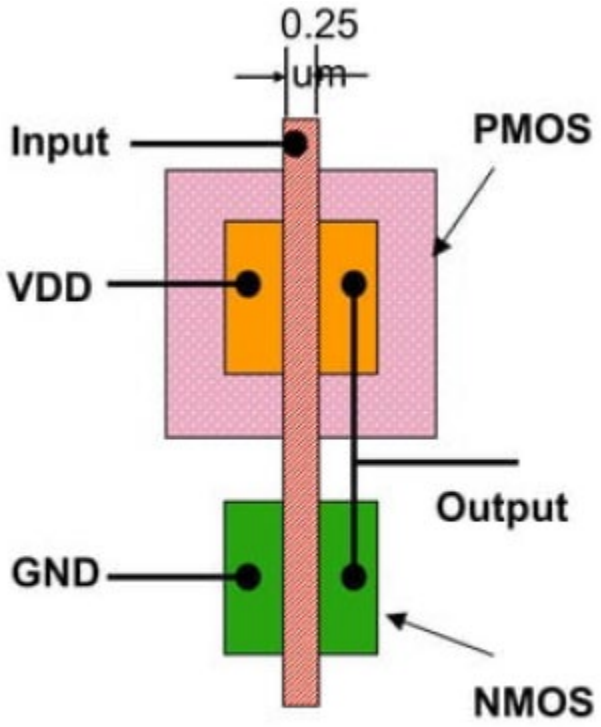
Gate Schematic



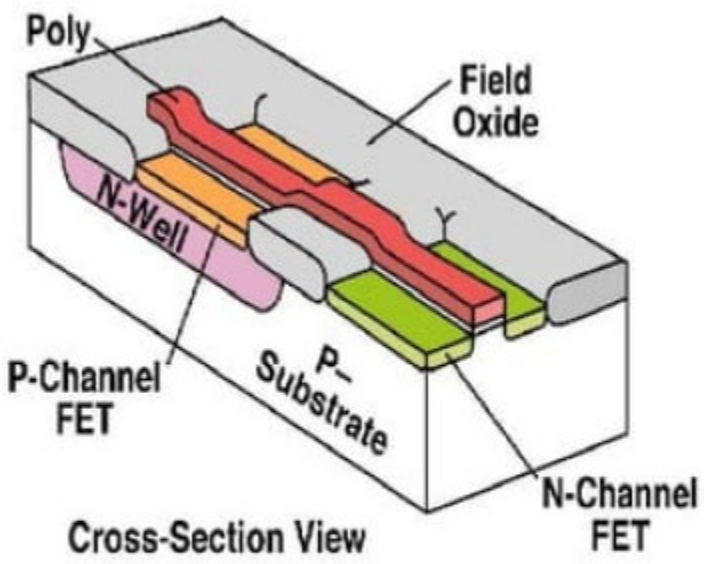
Transistor or Device View



Physical or Layout View



Aerial or Layout View



Wafer Cross-sectional View

1.2 ASIC (Application-Specific Integrated Circuit)

ASIC in VLSI stands for application-specific integrated circuit. This integrated circuit is aptly named since an ASIC microchip is designed and manufactured for one specific application and does not allow you to reprogram or modify it after it is produced. This means ASICs are not intended for general use. You must have ASICs created to your specifications for your product.

1.2.1 Types of ASIC

Gate array ASICs: Gate array ASICs offer the lowest level of customization. These ASICs start out with standard, predefined silicon layers. The only opportunity for customization is through manipulating the interconnections between transistors in the metallization stage of manufacturing. By opening and closing certain switches, you can achieve the desired function. However, gate array ASICs are limited in the functions they can perform.

Standard cell ASICs: Standard cell ASICs are semi-customizable to a greater degree than gate array ASICs. In standard cell ASICs, the silicon layers are comprised of library components, also called functional standard blocks. You can customize to match your specific function.

Full custom design ASICs: Fully custom ASIC designs allow you to create your ASIC from scratch, down to the transistor level, so you can make it to your exact specifications. This is extremely valuable for applications where standardized options are too limiting.

1.2.2 ASICs Applications

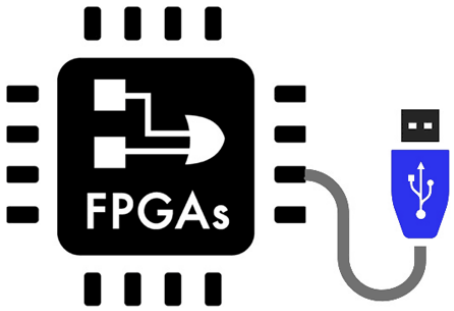
ASIC chip technology has a wide array of valuable applications. Generally, engineers use ASICs in products that are intended for permanent applications since they aren't designed to be modified. This includes electronic devices like smartphones, computers, voice recorders, and TVs, for example. There is virtually no limit to the types of applications for specific integrated circuits.

1.3 FPGA (Field-Programmable Gate Array)

A Field-Programmable Gate Array is an integrated circuit silicon chip which has array of logic gates, and this array can be programmed in the field i.e., the user can overwrite the existing configurations with its new defined configurations and can create their own digital circuit on field.

The FPGAs can be considered as "Lego Blocks" where the user has full control to customize the internal design of the FPGA by altering its functionality on hardware level to achieve its desired output.

FPGAs do nothing by itself whereas it is up to designers to create a configuration file often called a **bit file for the FPGA**. The FPGA will behave like the digital circuit once it is loaded with a bit file.



1.3.1 FPGA vs Microcontroller

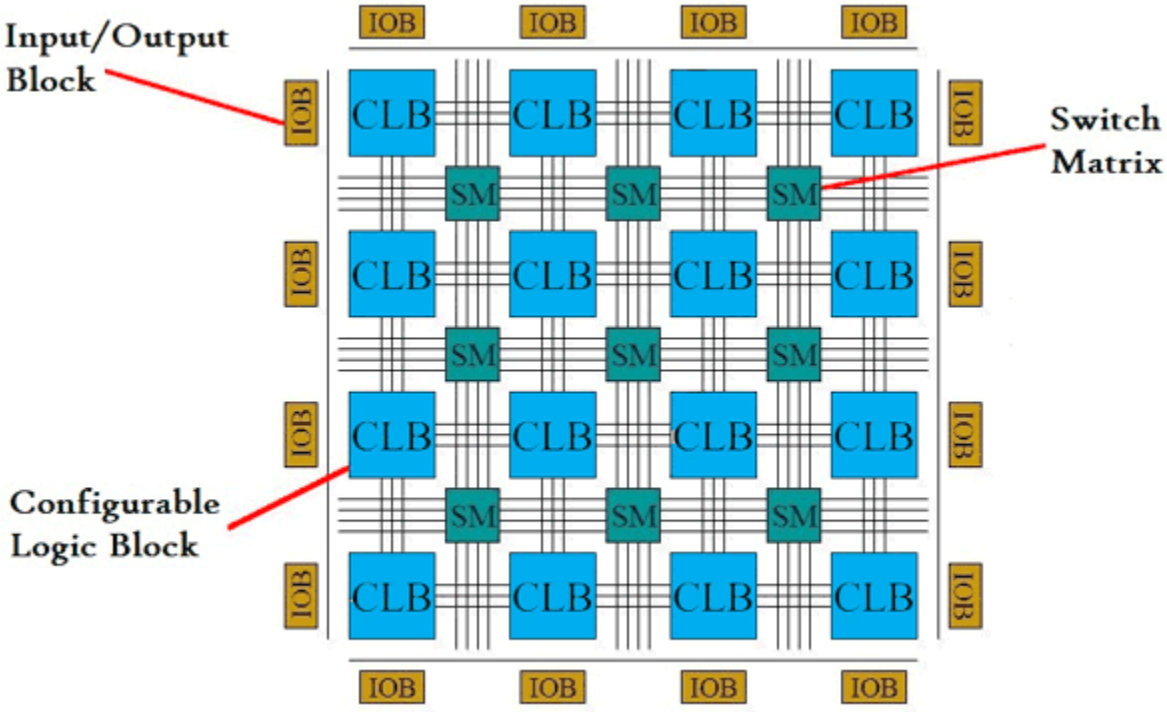
Microcontrollers cannot be programmed or restructured in the field. The user is neither allowed to overwrite its existing hardware configurations nor can they create any digital circuit on field. The microcontrollers are easy to program, and the community is also wide. The microcontrollers are custom built minicomputers which comes in IC form while FPGAs are only contains logic blocks that can again be rewired electrically.

Also, in terms of microcontrollers, it consumes less power than FPGAs. The FPGAs is known to be costly, and it requires more cost than microcontroller when it comes to building any device. FPGAs takes considerably much more time to set-up while the microcontrollers are available readily built for specific applications.

1.3.2 FPGA Architecture

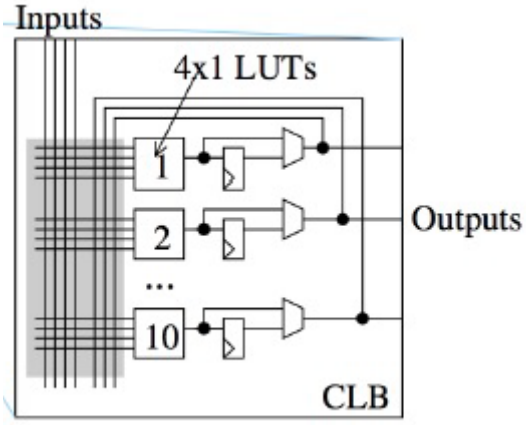
An FPGA has a regular structure of logic cells or modules and interlinks with each other (which is under control of the developers and designers). The FPGA is built with mainly three major blocks:

- Configurable Logic Block (CLB)
- I/O Blocks or Pads
- Switch Matrix/Interconnection Wires



CLB (Configurable Logic Block): These are the basic cells of FPGA. It consists of one 8-bit function generator, two 16-bit function generators, two registers (flip-flops or latches), and reprogrammable routing controls (multiplexers). The CLBs are applied to implement other designed function and macros. Each CLBs have inputs on each side which makes them flexile for the mapping and partitioning of logic.

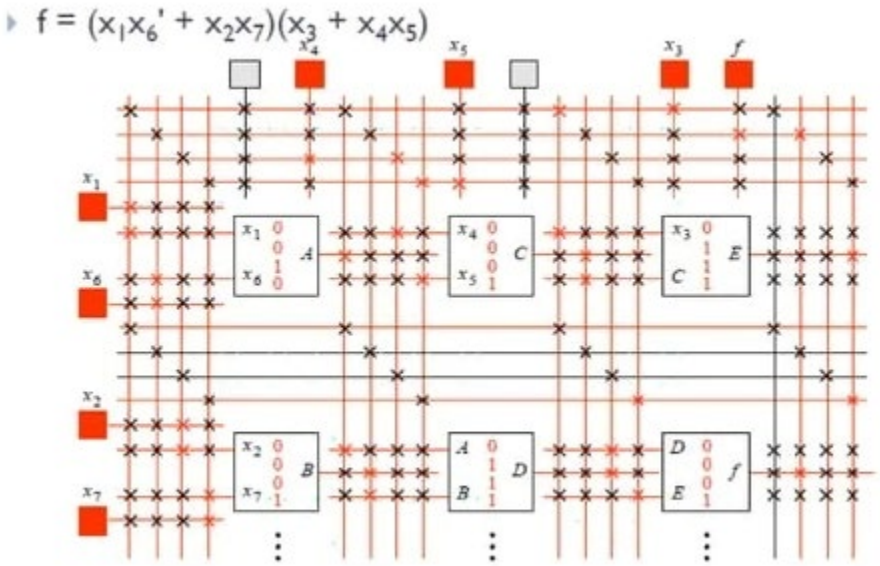
The CLB consists of LUT (Lookup Table, Gates, and Flip-Flop units):



I/O Pads or Blocks: The Input/Output pads are used for the outside peripherals to access the functions of FPGA and using the I/O pads it can also communicate with FPGA for different applications using different peripherals.

Switch Matrix/Interconnection Wires: Switch Matrix is used in FPGA to connect the long and short interconnection wires together in flexible combination. It also contains the transistors to turn on/off connections between different lines.

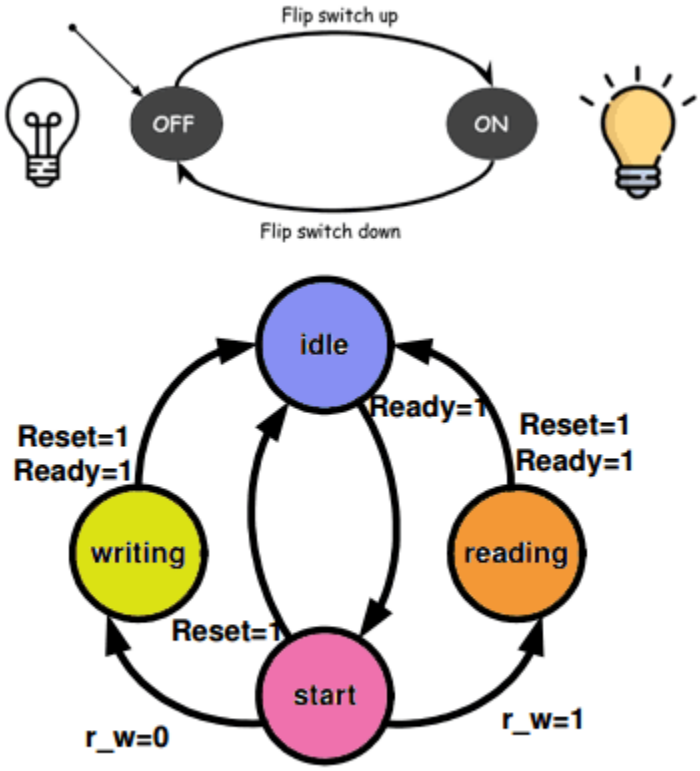
Example logic programmed on FPGA:



1.3.3 When FPGAs are Needed?

As mentioned above that microcontrollers have some limitation and cannot be used to perform task in parallel as microcontroller and microprocessors runs on sequential execution of programs which makes it a bit slow in some applications, in this scenario the FPGAs has an advantage and can be effectively used. Also, microcontroller can perform limited tasks because they come with instructions and their circuitry. A programmer has to abide by the restrictions while developing code. So, in this scenario also, the FPGAs have advantage.

However, in the case of microcontrollers, the processor switches from one code to another to achieve some level of parallelism. You will find it easier to write codes on microcontrollers than FPGAs. The parallel processing capability of FPGAs enables you to control interruptions effectively by using Finite State Machines (FSMs).



Finite State Machines (FSMs)

In the case of microcontrollers, you have to account for the time taken by ISR to resolve an interruption. You can rewire an FPGA easily just by reprogramming it. The configuration in an FPGA is loaded on the configurable logic cells when the power is switched on.

You don't need to make any changes (e.g., re-etching or re-fabrication) in the hardware to reprogram the FPGA. FPGAs are suitable for high-speed processing of parallel data and comes with a high degree of customizability. However, they also have the drawbacks of prototype operation and complexity of configuration.

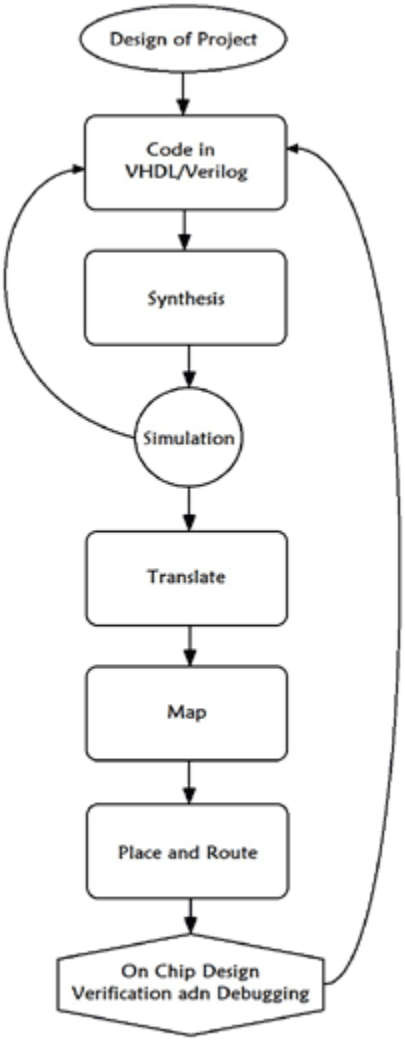
1.3.4 Programming of FPGA

Programming of FPGAs is done by **HDLs** (Hardware Description Languages). There are several HDLs are available, but the VHDL and Verilog are widely used HDLs.

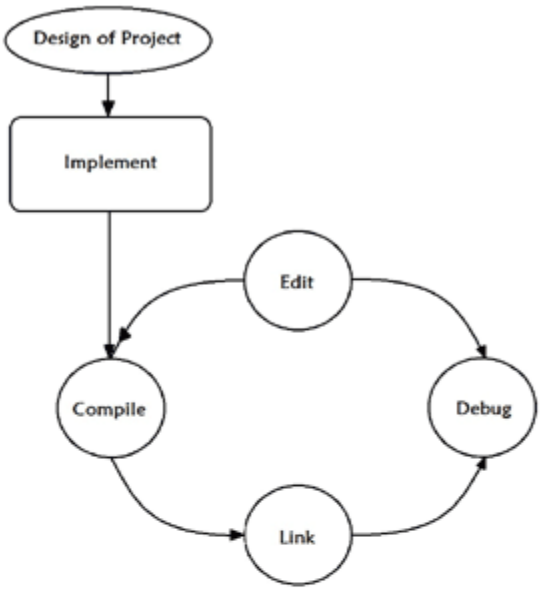
Even though there are some similarities between HDL code and high-level software programming language but the two are fundamentally different:

- Software codes are a sequence of operations and perform the processing in sequence.
- HDL code is a schematic that uses text to introduce components and create interconnections with parallel processing

To understand it in a better way, notice the difference between steps involved in Hardware and Software Design Flow in FPGAs and Microcontrollers respectively.



Hardware Design Flow



Software Design Flow

The Hardware design flow is used to program FPGAs whereas Software Design Flow is used to program the typical Microcontrollers and Microprocessors.

The important steps involve in programming FPGAs are as follows:

Synthesis: The First step is the synthesis which takes HDL code and translate into netlist which is a textual description of a circuit diagram or schematic.

Simulation: After synthesis, the next step involves the simulation which is used to verify if the design specified in the netlist functions correctly.

Convert netlist into Binary Format: Once the design is verified, the next is convert netlist into binary format. The components and connections are mapped to CLBs, and the design is placed and routed to fit onto the target FPGA (i.e Place and Route).

Perform Second Simulation: To see the design quality, a second simulation is performed.

Generate Bit File: Finally, a bit file is generated to load the design onto FPGA (A .bit file is a configuration file which is used to program all of the resources within FPGA).

Verify and Debug: At last, using different tools the design is verified and debugged while it is running on the FPGA.

Unlike the Hardware Design Flow, there is no requirement for pre-implementation simulation step in Software Design Flow. Also, the compile times for software are much shorter than implementation times for hardware designs so it is practical to recompile code and perform debugging as an iterative process.

1.3.5 The Programming Languages and Tools for FPGA

The most widely used are programming languages and tools are **VHDL** and **Verilog**. Both VHDL and Verilog are well established and wide support HDLs.

In terms of program FPGA, we need to forget the software coding behavior and start thinking about logic gates and circuits to implement the functionality that one wants to run on FPGAs.

There are many FPGA dev tools available such as:

VHDL/Verilog: Both languages provide structures to describe the inherently parallel nature of FPGA / ASIC development. Due to their initial use to describe the behavior of the circuits prior to the generation of synthesis tools, these languages also support test benches to test the design being implemented.

LabVIEW FPGA: The LabVIEW is a graphical language which gives a completely different way of programming a FPGA. LabVIEW FPGA is the FPGA compilation uses a cloud-based option, which speeds up the compilation time significantly.

MATLAB: The MATLAB is the language which can play a vital role and should be studied. The MATLAB is generally used to generate filters for signal processing, develop image processing algorithms and almost any other algorithm. But apart from this, it is possible to go from MATLAB model to FPGA using the HDL coder. The traceability enables the high integrity applications can be developed using this approach. HDL coder enables to perform hardware (FPGA) in the loop testing and co-simulation to see the difference between the original algorithm and the implemented hardware algorithm, which helps to explore the design space.

There are other tools such as MyHDL, JHDL, BSV, System Verilog, SPINAL HDL, CHISEL, C/C++/System C etc.

1.4 RISC-V ISA Architecture

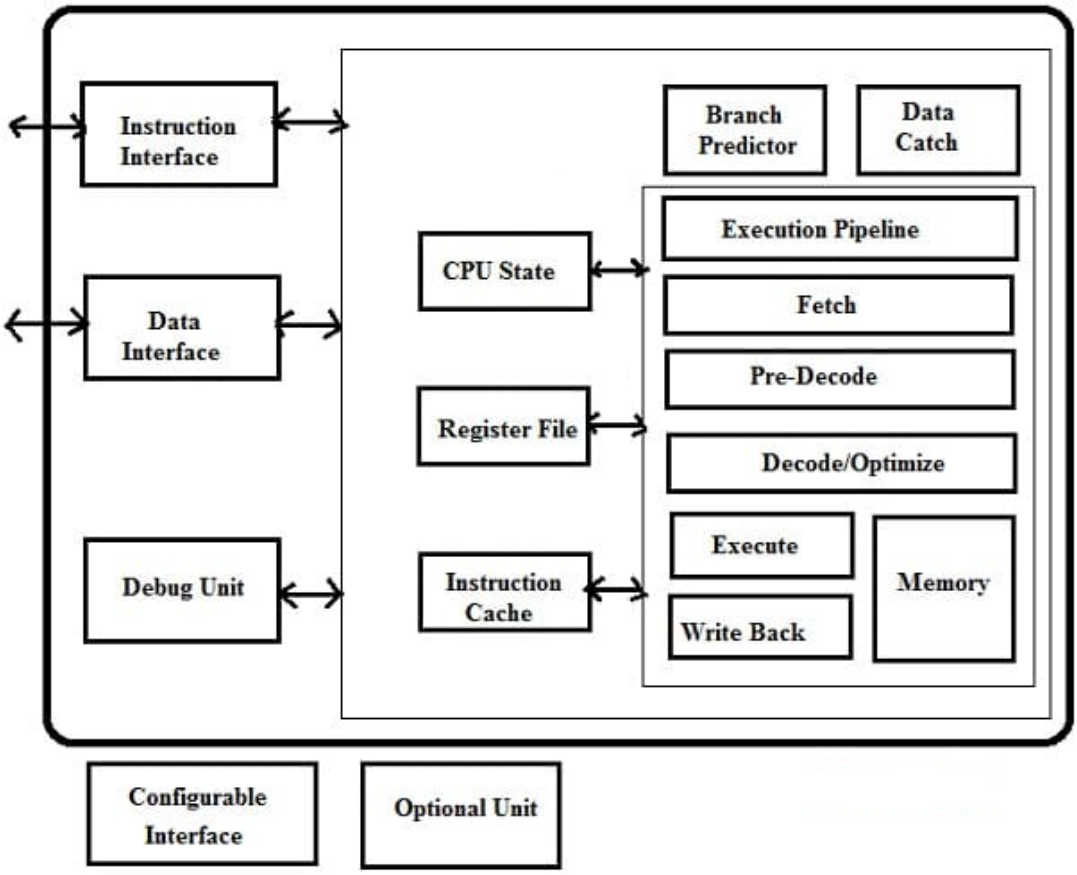
RISC V is an "Instruction Set Architecture" (ISA) developed by the University of California, Berkeley since 1981 and became available in 2019. The concept of RISC was motivated by the truth that most of the processor instructions were not utilized by most computer programs. So, unnecessary decoding logic was being utilized within the designs of processors, consuming more power as well as area. To shorten the instruction set & to invest more within register resources, the RISC V processor was implemented.

This technology is completely open source & free so anyone can make their new processor designs using RISC-V ISA. And can also modify and extend the RISC-V ISA according to his requirements.

In the RISC V processor, the term RISC stands for "reduced instruction set computer" which executes few computer instructions whereas 'V' stands for the 5th generation. It is an open-source hardware ISA (instruction set architecture) based on the established principle of RISC.

1.4.1 RISC V Architecture & Working

The RV12 RISC V architecture is shown below. The RV12 is highly configurable with a single-core RV32I and RV64I compliant RISC CPU which is used in embedded fields. The RV12 is also from a 32 or 64-bit CPU family depending on the industrial standard RISC-V instruction set.



The RV12 simply executes a **Harvard architecture** for simultaneous access to instruction as well as data memory. It also includes a 6-stage pipeline which helps in optimizing overlaps in between the execution as well as memory accesses to improve efficiency. This architecture mainly includes Branch Prediction, Data Cache, Debug Unit, Instruction Cache, & optional Multiplier or Divider Units.

1.4.2 The RISC V Execution Pipeline

It includes five stages:

- IF (instruction fetch)
- ID (instruction decode)
- EX (execute)
- MEM (memory access)
- WB (register write-back)

Instruction Fetch: In Instruction Fetch or IF stage, a single instruction is read from the program counter (PC) and instruction memory which is updated to the next instruction.

Instruction Pre-Decode: The Instruction Pre-Decode stage will decode a 16-bit-compressed instruction into a native 32-bit instruction.

Instruction Decode: In the Instruction Decode (ID) stage, the Register File is allowed & the bypass controls are decided.

Execute: In Execute stage, the result is calculated for an ALU, DIV, MUL instruction, the memory allowed for a Store or Load instruction, and branches & jumps are measured against their expected outcomes.

Memory: In this Memory stage, the memory is accessed through the pipeline. The inclusion of this phase ensures the pipeline's high performance.

Write Back: In this stage, the Execution stage result is written into the Register File.

1.4.3 Difference between RISC-V vs ARM

The difference between RISC-V Vs ARM includes the following:

RISC-V	ARM
RISC-V is open source, so it doesn't require any license	ARM is a closed source, so it needs a license
It is a new processor platform, so there is very small support for software & programming environments	ARM has a very large online community, which supports libraries & structures to assist the target designers in various platforms like microprocessors, microcontrollers & also servers
RISC V-based chips use up to 1 watt of power	ARM-based chips use up to 4 watts of power
It has a fixed & variable ISA encoding system	It has a fixed ISA encoding system
RISC-V instruction set size ranges from 16-bit to 128-bits	Its instruction size ranges from 16-bit to 64-bits
It includes 32 general-purpose & floating-point registers	It includes 31 general-purpose & floating-point registers
It has 26-single precision floating point operations	It has 33-single precision floating point operations
It has 26-double precision floating point operations	It has 29-double precision floating point operations

1.4.4 Advantages of RISC-V

The advantages of the RISC V processor include the following:

- By using RISCV, we can save development time, software development, verification, etc.
- This processor has many pros like simplicity, openness, modularity, clean-slate design, and extensibility.
- This is supported by several language compilers like the GCC (GNU Compiler Collection), a free-software compiler & through the Linux OS.
- This can be used by companies freely due to no royalties, no licensing fees & no strings connected.
- RISC-V processor doesn't include any new or innovative features because it simply follows established principles of RISC.
- Similar to several other ISAs, this processor specification simply defines various instruction set levels. So, this contains 32 & 64-bit variants as well as extensions to give support for floating point instructions.
- These are free, simple, modular, stable, etc.

1.4.5 Disadvantages of RISC-V

The disadvantages of the RISC V processor include the following:

Complex instructions are frequently used by compilers & programmers.

These processors need to save a variety of instructions rapidly, which requires a big cache memory set to respond to the instruction within a timely manner.

The complete features, capabilities & benefits of RISC mainly depend on the hardware architecture of CPU.

1.4.6 Applications of RISC-V

The applications of the RISC V processor include the following:

RISC-V is used in embedded systems, artificial intelligence & machine learning.

These processors are used in high-performance-based embedded system applications.

This processor is appropriate to use in some particular fields like edge computing, AI & storage applications.

RISC-V is important as it permits smaller device manufacturers to design hardware without paying a lot of money (to ARM, Intel, e.g.) to get license to use their IAS in their CPUs.

This processor simply allows the researchers and developers to design as well as research with a freely available ISA.

The applications of RISC-V range from small embedded microcontrollers to desktop PCs & supercomputers including vector processors.

1.4.7 RISC-V International Members

Google, HUAWEI, Intel, ZTE, Qualcomm, SEAGATE, Western Digital, AMD XILINX, Arduino, ESPRESSIF, IBM, Microchip, Alibaba Cloud, NOKIA, NVIDIA, Raspberry Pi, SIEMENS, SONY, UET Lahore, NUST, UET Taxila, UET Peshawar, etc.