EuroCDP
EU Chips Design Platform

Open-source PDKs and EDA tooling recommendations

May 2026

Authors: Joonas Multanen, Laxmi Karkee, Pekka Jääskeläinen, Macarena Martínez-Rodríguez, Dzmitry Pustakhod, Krzystof Herman, Diego Gigena-Ivanovich, Emanuele Bottino, Francesc Serra-Graells, Caaliph Andriamisaina

Disclaimer

The European Commission is not liable for any use that may be made of the information contained herein.

Copyright notice

© 2026 EuroCDP

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Executive Summary

This document provides an overview of open-source process design kits (PDK) and electronic design automation (EDA) tooling status in the domains of digital, analog, mixed signal, radio frequency (RF) and photonics design. A typical flow in each domain is first described, after which existing open-source tools are listed and comparisons of their features are presented. An analysis pointing out the major gaps between open-source and commercial tools is presented, and finally, indicative recommendations on tools to use are proposed.

The goal is to provide designers with guidelines on the capabilities of open-source EDA tools compared to existing proprietary solutions. This should help designers decide whether open-source EDA tools are appropriate for their projects. On the other hand, the gap analysis should provide recommendations for improving the open-source EDA tool chain.

Since 2020, designers have had the option to tape out designs with open process design kits (PDK), where the process technology information is openly available. Today there are three manufacturers (of which one is based in the EU) offering manufacturable open PDKs. Currently, up to 130 nm process nodes are offered.

The maturity of the different design flows varies. Digital design tools seem to have been developed enthusiastically by the community in recent years, making it the most mature of the analyzed flows. While there aren't apparent gaps that would prevent producing functioning chips with open-source tools, there is room to improve the support for advanced process nodes and features such as design for test (DFT). Hands-on experimentation revealed that design timing and area consumption produced by open-source physical design tools varies: In one design case the produced results were on par with commercial tools, whereas another case resulted in notably worse results.

In the analog design flow, there is currently a toolchain available which allows analog IC design based on an open PDK. In contrast, for mixed signal design, major gaps still exist, with no clearly established de facto tools. These gaps include insufficient support for advanced analysis capabilities, as well as usability and performance issues of the simulators.

The open-source RF design ecosystem is quite young and adapts previously existing EDA tools for its purposes. Many capabilities found in commercial software are yet to be implemented in open-source tools, but there seems to be continuous development in this domain.

In the photonics flow, the open-source script-based tooling supports the layout implementation quite well. The integration of these tools with schematic editors and circuit-simulators is missing: Currently, there are no open-source design environments to connect the different design steps and tools. In the future, there is a need to enable this to cover more advanced use cases. Another area to improve is the standardization of file formats and interfaces between the tools.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Table of Contents

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Glossary

Definition	Abbreviation
Assembly Design Kit	ADK
Application Specific Integrated Circuit	ASIC
Application Specific Instruction set Processor	ASIP
Clock Tree Synthesis	CTS
Design For Testing	DFT
Design Rule Check	DRC
Electronic Design Automation	EDA
EigenMode Expansion	EME
Hardware Description Language	HDL
Integrated Circuit	IC
Instruction Set Architecture	ISA
Layout versus Schematic	LVS
Photonic Integrated Circuit	PIC
Power, Performance and Area	PPA
Process Design Kit	PDK
Process, Voltage, Temperature	PVT
Quality of Results	QoR
Register Transfer Level	RTL
Rigorous Coupled Wave Analysis	RCWA
System Verilog	SV
Test Design Kit	TDK
Transaction Level Modeling	TLM
Transfer Matrix Method	TMM
Universal Verification Methodology	UVM

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

1. Design Flows

Electronic Design Automation (EDA) refers to the use of advanced software and tools to design, simulate, and verify electronic systems such as integrated circuits (ICs), Application Specific Integrated Circuits (ASIC) and Systems-on-Chip (SoCs), including processors and peripherals. Since modern ASICs consist of billions of transistors, manual design is impractical, and automation tools are essential. Without EDA, the development of Very Large-Scale Integrated Circuits (VLSI) would be impossible. ASIC design involves creating complex electronic systems tailored for fabrication as specialized semiconductor devices, which are typically used in high volume products or applications with strict power, performance and size requirements.

This section describes typical flows for five domains of semiconductor design where EDA tools are involved: digital, photonics and analog, mixed-signal and radio frequency (RF) design. This section is then used as a base when open-source EDA design tools are introduced in later sections.

1.1. Digital Design

Digital circuit design relies on EDA tools at every stage, from the transistor level to system-level architectural exploration. The design of a digital system often begins with system-level architectural exploration, where designers evaluate different architectures and performance tradeoffs before detailed implementation. This stage is especially critical for designing complex, high-performance computing systems such as graphics processors and server-class CPUs. Once the architecture is defined, typically a Hardware Description Language (HDL) is used to describe the hardware implementation. The implementation is then taken through a backend physical design flow. Through a sequence of design, verification, and optimization stages, this abstract description is gradually transformed into a physical layout. The final deliverable is a GDSII file, the industry standard format used by semiconductor foundries to manufacture chips.

Digital design process includes various steps like design conceptualization, chip optimization, logical and physical implementation, and design validation and verification. The overview of VLSI ASIC design flow is shown in the figure below.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

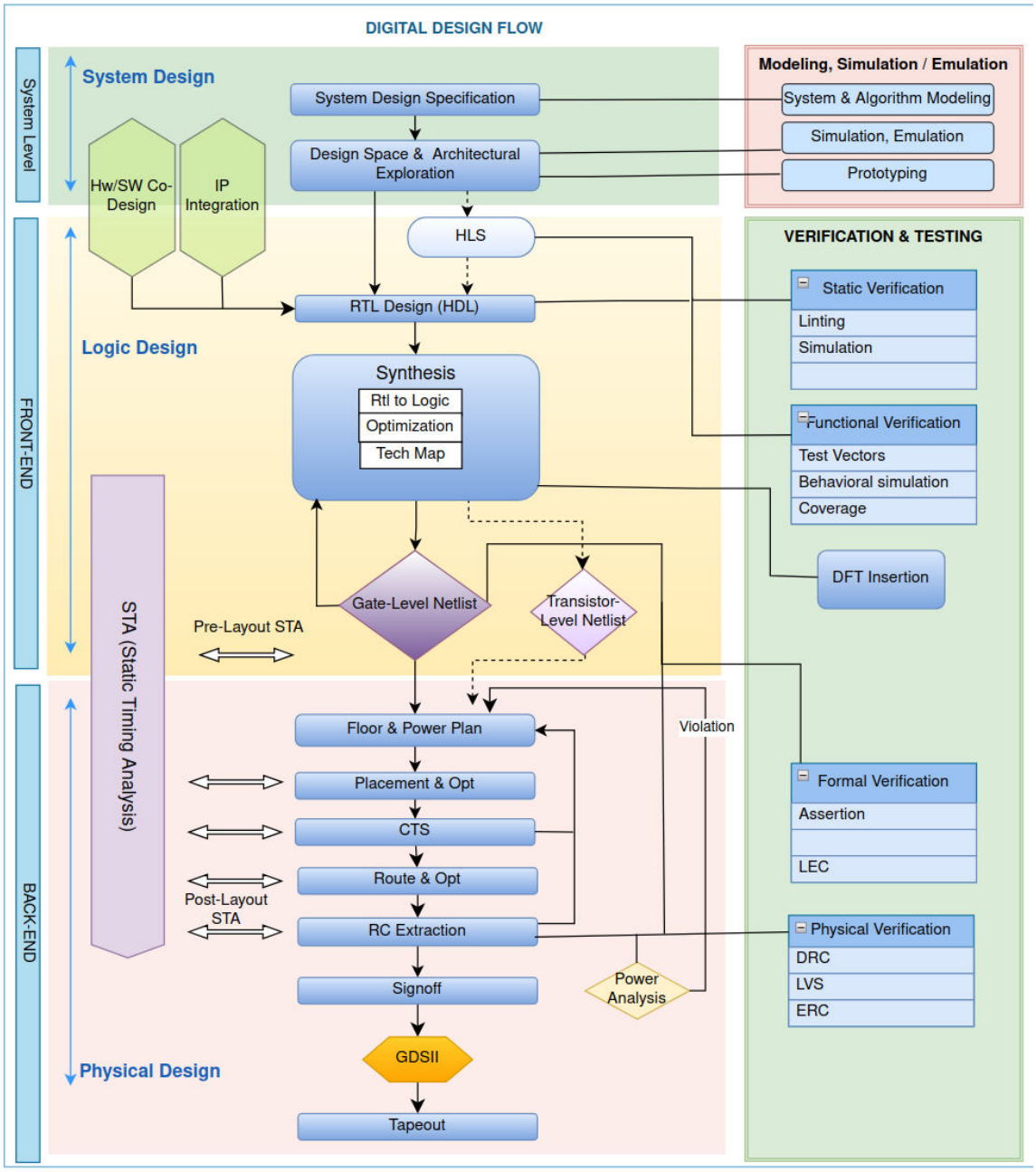

Figure 1. Overview of Digital Design Flow.

In general, chip design is broadly divided into front-end design consisting of logic design, including RTL development, functional verification, and synthesis, and back-end design, which consists of physical implementation processes such as floorplan, placement, routing, static timing analysis and sign off. While some domains such as static timing analysis and verification often overlap, physical design tasks are generally classified as back-end. As technology continues to advance, achieving high-speed performance and energy efficiency has become increasingly important in ASIC design. To handle the growing complexity of modern ASICs and their implementation, a top-down design approach is commonly used.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

1.1.1. System Level Design

Beyond defining specifications, the objective of system level design is to explore design opportunities, identify architectural trade-offs, and make informed decisions to ensure the system satisfies both functional requirements (correctness, performance) and non-functional requirements (power consumption, silicon area, cost, programmability, safety, reliability). A critical activity in this phase is architectural exploration, where multiple architectural alternatives are modeled and evaluated to determine the most suitable implementation strategy before proceeding with Register Transfer Level (RTL) implementation. This helps optimize performance, reduce risk, and minimize rework in later stages of the flow. To manage the complexity of ASICs, system design typically uses various methods, modularity and hierarchical models, and progresses through several abstraction layers, which are described in subsequent sections.

System Specification

The development of a detailed and accurate specification establishes a solid foundation for complex ASIC design. The specification defines the chip's intended behavior, functionality, and interfaces, along with the overall architecture at a high level. All requirements must be clearly specified to ensure correct implementation. These requirements guide the overall design process, and typically the architectural exploration in the next stage involves evaluating and modeling different design candidates at varying levels of accuracy. At this stage, the architecture for technical feasibility is analyzed, and the design challenges, operational constraints, and environmental conditions are thoroughly assessed. Architectural trade-offs are analyzed to balance performance, power, area, cost, and implementation complexity, and any required possible third-party IP blocks are identified.

Architectural Design

At this stage, high-level decisions are made regarding system organization, functionality, performance objectives, and key architectural trade-offs. The architecture defines the required system components, clock frequencies, power constraints, and overall functional behavior. The system-level design is then decomposed into subsystems, functional blocks, and modules, with their interfaces and high-level data flow formally described, and memory hierarchies, clocking strategy, interfaces, communication protocol and data paths are designed. The design process forms a system-level view which includes the following:

- **Component requirements:** Identification and selection of compute elements such as CPUs, GPUs, accelerators, memory hierarchy, caches, interconnects, and peripheral blocks.
- **Clock frequencies:** Determination of operating frequencies for subsystems to achieve performance goals.
- **Power, performance and area (PPA):** Early estimation of power consumption, throughput, latency, and die area to ensure design feasibility.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

- **Data flow:** Definition of how data moves between subsystems and functional blocks within the chip.

System Level Modeling, Simulation and Emulation

At the architectural design and exploration phase of ASIC design, division of tasks to be executed on hardware and software blocks is considered. System level modeling starts at higher abstraction with abstract modeling of behavior and platform which is iteratively refined in subsequent steps in top-down manner towards final implementation when required. Designs can be modeled at various abstraction levels such as Transaction Level Modeling (TLM) or Register Transfer Level (RTL). Model based approach begins with application first, enabling estimation of feasibility of implementation alternatives at an early stage of system design. Model based method increases reusability of components, reducing development time where HW and SW can be co-designed. System-level simulation, modelling, and exploration of the design are typically performed using high-level programming languages such as C++ or SystemC. Additionally, initial architecture can be modeled using HDL to represent the intended behavior of the ASIC.

Transaction Level Modeling (TLM): TLM is designed for early SoC exploration within the design flow, offering a lightweight development effort at higher abstraction level above RTL. In contrast, RTL implements a cycle-accurate HDL description for timing analysis resulting in slower simulation speeds and usually available only in later design stages. TLM representation of hardware blocks or IPs provides bit-accurate functionality, register-accurate interfaces, and system-level synchronization. Timed and untimed TLM address different design needs within a common modeling framework. Untimed TLM is used in early architectural exploration for software development, and functional verification, where fast simulation is crucial compared to timing accuracy. However, timed TLM adds timing annotations to capture micro-architectural and communication behavior, enabling more accurate simulation for real-time systems and architectural evaluation. Key advantages of TLM include:

- Early software development in HW/SW co-design
- Architectural analysis
- Efficient functional verification
- Virtual software development

SystemC: SystemC is a system-level design and modeling language developed to accelerate complex HW/SW co-design while meeting system requirements and performance targets. It enables both hardware and software to be modeled within a single, unified language. SystemC is built on C++ and provides an object-oriented framework through a standardized set of classes for system modeling and simulation.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

As design complexity grows, simulation time increases significantly, hence hardware-based debugging approaches such as emulation and (physical or virtual) prototyping are critical for large-scale and complex ASIC development. Emulation offers higher runtime performance and improved debugging efficiency, making it suitable for early RTL functional verification of complex SoCs. In emulation, HDL designs (eg. Verilog or VHDL) are compiled and mapped onto specialized hardware platforms which closely reflect real hardware behavior. This allows execution at much higher speeds, typically in hundreds of KHz or MHz range compared to functional simulations. Emulators combine specialized hardware and software components, providing an effective solution for high-speed design verification and debugging.

HW/SW Co-design

Hardware–software co-design refers to the integrated development of both hardware and software components to produce a complete system. This collaborative approach helps designers meet performance and functional requirements while improving design quality, shortening development cycles, reducing cost, and managing the growing complexity of modern SoC architectures. It is an iterative process that enables rapid early prototyping to validate specifications and timely feedback during the design cycle. With the increasing complexity of ASIC and SoC designs considering computer architecture and software compilation, HW/SW co-design becomes inherently challenging.

Transforming a system-level specification into a mixed HW/SW design involves system specification, communication synthesis, and architecture generation. Architecture generation yields a set of hardware and software modules and includes two tasks: virtual prototyping model which can be simulated, and architecture mapping, which produces an implementation or emulation of the original specification.

IP generation, Reuse and Integration

Intellectual Property (IP) are reusable design blocks that are either developed in-house or licensed from third-party vendors which can be integrated into designs. These IP blocks may range from simple peripheral controllers such as UART and SPI to complex subsystems including CPUs, GPUs, memory controllers, RAM macros, high-speed interfaces, and specialized accelerators. With increasing design complexity, scale, and shorter development cycles, the reliance on pre-verified and reusable IP has grown significantly. IP is typically delivered in multiple forms including Soft IP consisting of synthesizable RTL code, Hard IP that is delivered as fully placed and routed with process-specific GDSII or OASIS layout, and Firm IP which provides an optimized netlist but without a complete physical layout.

IP reuse offers clear advantages, including reduced time-to-market, lower development cost, minimized design risk, and access to specialized functionality. However, integrating heterogeneous IPs on a single chip introduces challenges related to interface compatibility, differing design methodologies, and strict design rules at advanced process nodes. To address these issues,

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

established methodologies, best practices, and EDA assisted IP generation, and integration techniques are widely adopted.

IP integration involves selecting suitable IP based on system requirements, validating its functionality within the full-chip environment, adapting interfaces through wrappers if needed, configuring parameters, and performing physical integration, particularly for hard IP by placing and connecting the blocks within the layout.

1.1.2. Logic Design and Synthesis (Front-End)

Once the system architecture and block-level planning are complete, the process continues with design implementation. This process starts with the front-end, where the HDL designs are generated and transformed from a high-level system description into a target technology mapped description. The hardware descriptions can be generated by high-level synthesis (HLS) tools, written manually in HDL, or integrated as pre-existing IP blocks.

High Level Synthesis (HLS)

A method of generating HDL designs from behavioral specifications is known as high-level synthesis. In HLS, the system's behavior is described in high level languages such as C, C++, or Python which is easier to understand and write compared to low level HDL code in VHDL. This accelerates the design process and reduces the likelihood of errors. During HLS, high level description is analyzed then data flow and operations of the system are extracted and mapped to equivalent RTL that describes the system in terms of registers, operations and data flow.

In FPGA, HLS also enables the description and implementation of parallel hardware systems using high-level parallel programming models. Standards such as OpenCL provide an abstraction for expressing Single Program Multiple Data (SPMD) computation, in which a single behavioral description executes simultaneously across multiple data elements. By allowing parallel hardware to be specified in high-level languages rather than low-level HDL, this approach permits synthesis tools to compile the behavioral description into hardware structures that realize the intended parallelism while improving design productivity.

RTL Design (HDL Coding)

RTL design is a methodology for designing digital circuits that serves as an abstraction representing data flow between hardware registers and data processing or manipulation using arithmetic and logical operations. RTL design stage is where high-level specifications of a chip are translated into hardware implementation typically in the form of HDL such as VHDL, Verilog or SystemVerilog that describes the intended functionality of ASIC which includes modules, registers, combinational gates (AND, OR, NOR, NAND) as well as sequential logic (Flip-Flops, latches), Finite State Machine (FSM), Arithmetic Logic Block, Datapath and other elements.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

At the RTL coding stage, it is essential to ensure that the HDL implementation correctly reflects the intended system behavior and satisfies all design requirements. This may include iteratively optimizing the code for PPA and ensuring that it remains modular and reusable. During and after the implementation, the RTL is verified, which will be discussed in more detail in later sections.

RTL partitioning is an effective technique used to break down the complexity of large VLSI/ASIC designs into smaller, manageable modules or subsystems. This approach not only simplifies the design process but also enhances design quality by enabling focused development on each individual component. In addition, partitioning supports parallelism, allowing different blocks to be designed and verified independently and concurrently. A design may be partitioned in several ways:

- **Functional partitioning,** based on major functional units such as processor cores, memory blocks, and I/O interfaces.
- **Hierarchical partitioning,** which organizes the design across multiple abstraction levels.
- **Physical partitioning,** which groups components according to physical characteristics, such as power domains or clock domains.

RTL Optimization

The goal of RTL optimization is to improve the design's PPA by refining the RTL description to achieve greater efficiency while meeting the design requirements. This process requires an understanding of the design as well as the optimization techniques used to improve PPA. Typically this involves trade-offs of the different optimization targets which are:

A. ***Performance optimization*** focuses on improving the operating speed of the design. This can be achieved through techniques such as pipelining, that divide design into various stages for parallel execution and simultaneous multiple operations.

B. ***Power optimization*** aims to reduce the overall power consumption of the design. Techniques like clock gating accomplish this by disabling the clock signal for blocks that are not active, lowering switching activity of transistors and therefore reducing dynamic power.

C. ***Area optimization*** focuses on minimizing the silicon footprint of the design. This can be achieved through methods such as resource sharing, where multiple functions reuse the same hardware resources, as well as constant propagation and other techniques that eliminate unnecessary logic.

Synthesis

After the RTL implementation has been verified, the design proceeds to synthesis. Synthesis is the process where EDA tools translate the high-level RTL description into a gate-level netlist composed of technology specific standard cells that can be physically implemented on silicon.

During synthesis, the required logic gates and flip-flops needed to implement the intended functionality are identified and then transformed into a logical representation, producing a netlist

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

that defines the necessary interconnections. The synthesis tool uses RTL files, standard cell libraries containing timing and electrical data, constraint files, and physical abstracts such as .lef files that define cell dimensions and metal layers. It then transforms the behavioral or dataflow description into optimized physical standard cells, which are defined in a process design kit (PDK) containing the information required for fabrication on a target technology. Synthesis uses various optimization techniques to improve performance, power, and area efficiency. Information from the implementation process can be used iteratively, such as generating a placement for the standard cells and then rerunning the synthesis with the placement information, which can allow better decisions during synthesis. This method is known as physical-aware synthesis.

The synthesis flow consists of several stages, and its goals, categories, and typical inputs/outputs can be summarized as follows:

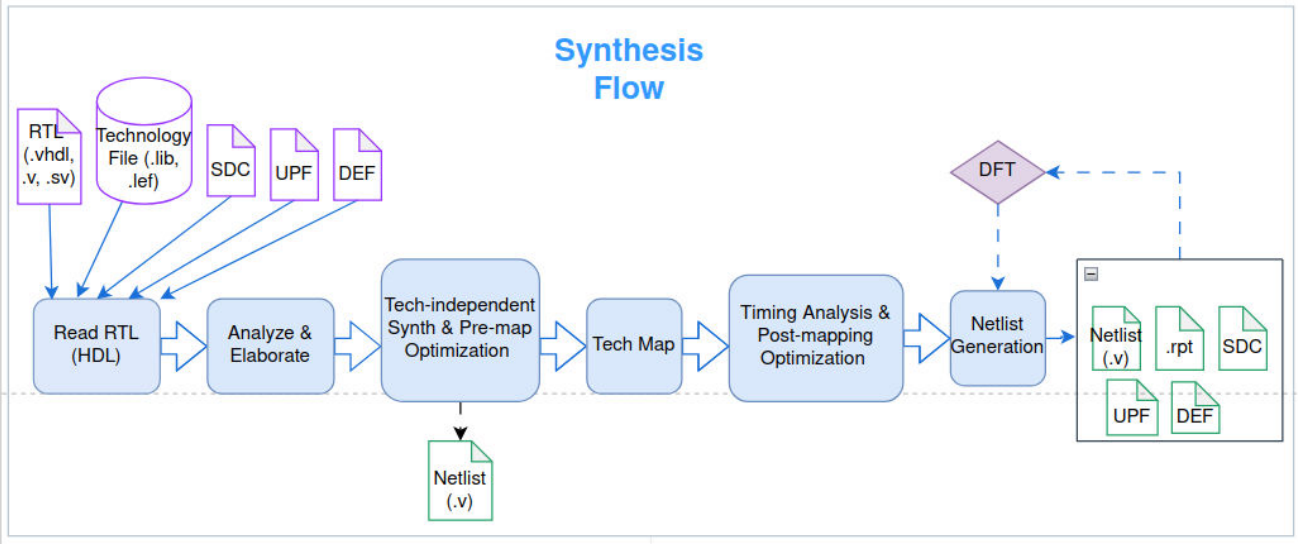

Figure 2. RTL Synthesis Flow.

Goal of synthesis

- Logic optimization with good QoR (Quality of Results)
- Scan Insertion (DFT)
- Netlist generation and verification with LEC (logic equivalence check)

Synthesis inputs

- **Liberty-timing library (.lib or .db):** Describe logical function, timing, power and area of standard cells including PVT (Process, Voltage, Temperature) characterization.
- **Physical library (.lef):** Library exchange format contains physical information like height, width of metal, via, pin, standard cells and macros.
- **SDC:** Timing constraints.
- **RTL:** Code in HDL format.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

- **DEF:** Contains placement information of macros, pre-placed cells, IO ports, block size and blockages mainly used for physical-aware synthesis.
- **UPF:** Provide information on power intent of design including power domain, level shifter, isolation cell and retention registers.

Synthesis output

- Netlist
- UPF (updated)
- SDC (updated)
- DEF (updated)
- Reports

Synthesis Flow

Synthesis flow typically includes following steps:

Analysis and elaboration: During elaboration, the tool verifies that the design is well-defined and unique, resolving high-level constructs for any unresolved references, such as hierarchical module instantiation, blocks generation, parameter overrides, and loop unrolling. After elaboration, timing loops are checked, and the synthesis tool builds a generic, technology-independent logic-level netlist without mapping it to actual standard cells.

Technology-independent synthesis and (pre-mapping) optimization: Prior to technology mapping, synthesis tools perform various high-level, technology independent optimizations that include constant propagation, eliminating dead code or unused modules, Boolean simplification, finite state machine (FSM) encoding and extraction and retiming.

Technology mapping: At this stage, elements such as gates and flip-flops are mapped to actual standard cells from the .lib file. Mapping involves selecting the most suitable cells for implementing required logic functions while considering factors like timing, area and power consumption. It includes various processes like decomposition which breaks complex expressions into smaller gates supported by the target PDK, cell selection based on appropriate drive strength to meet delay, area and power targets, buffer insertion to handle long wires or high fanout nets and MUX inference to implement conditional logic as optimized multiplexer structures.

Timing analysis & post-mapping optimization: Synthesis tools further refine the design through timing analysis and optimization techniques to ensure it meets the required constraints. This involves evaluating signal and interconnect delays along with other factors that affect overall performance. During this stage, the design is optimized according to specifications through methods such as register retiming to improve pipeline stage placement, cell resizing by replacing cells with higher or lower drive strengths to meet timing requirements, clock gating to reduce power consumption, gate duplication to manage high fan-out loads, and rebuffering to adjust or replace buffers based on updated net delays.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Netlist generation: Finally, synthesis generates a mapped and optimized gate-level netlist that will be used in physical design. The synthesized netlist not only represents the logic implementation but also includes attributes that influence how the design meets timing, area, and power goals. At the end of the synthesis process, additional reports and updated constraint files are also produced, depending on the synthesis tools and configurations.

1.1.3. Verification, Testing, Power and Timing Analysis

In the physical design of complex ASICs, simulation, verification, power, and timing analysis are critical activities that ensure a design meets its functional, performance and reliability requirements before manufacturing. Verification consumes a significant portion of chip design cycle, and it can be performed in various stages of the design flow including front-end and backend flow consisting of logic as well as physical design. Functional simulation and verification confirm correctness of logic and interfaces, while static timing analysis (STA) validates that timing constraints are met across operating corners. Power analysis evaluates dynamic and leakage power to ensure thermal and energy consumption are satisfied. Traditionally, many of these checks, along with physical verification such as DRC, LVS, and ERC, are performed during late-stage signoff, once the layout is relatively complete and prepared for tape-out.

Left-Shifting

As the size and complexity of modern SoCs increases, relying primarily on end-of-flow verification has become increasingly inefficient. Late discovery of timing, power, or reliability issues often leads to costly design iterations, extended schedules, and increased risk at tape-out. To address these challenges, new concepts are being adopted as a left-shift verification approach in the ASIC design flow. Left-shift moves selected verification, power, and timing analyses into earlier stages of the design flow, where issues can be identified and corrected more quickly and with lower impact. Rather than applying full signoff checks on incomplete layouts, left-shift emphasizes lightweight, targeted pre-simulation and pre-layout analyses that focus on critical risks such as power domain errors, clock and signal crossings, leakage paths, and interface mismatches. This strategy improves productivity, reduces late-stage rework, and enhances overall design quality while maintaining manageable verification complexity.

Modern verification combines multiple methodologies and abstraction levels to comprehensively validate a design.

Behavioral Simulation

Behavioral simulation uses high-level models such as behavioral RTL, SystemC, or C/TLM models to validate design functionality early in the flow. It focuses on algorithm correctness, data flow, and architectural intent, providing fast simulation for design exploration.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Functional verification

Following RTL design, functional verification ensures that the RTL design performs according to specifications focusing on logic correctness, data flow, and module interactions. This involves code analysis (linting), RTL simulation, and coverage analysis. This is typically achieved through simulation-based methods, where testbenches apply input stimuli and observe responses, mimicking hardware behavior. Functional verification is essential for detecting logic errors, interface mismatches, and integration issues. It includes various steps and methodologies:

Linting: Linting is used to verify RTL design correctness. Linting tools examine the RTL code for common coding mistakes, compliance with coding standards, and potential synthesis or timing issues. By identifying these problems early in the design process, linting helps improve RTL quality.

Assertion-based verification: RTL code assertions are used to automatically monitor and validate specific design properties, such as protocol compliance, signal integrity, and corner-case behaviors. Assertions complement simulation by providing automatic checks that reduce manual debugging effort.

Coverage-driven verification: Coverage analysis measures how much of the design has been exercised during verification. Functional and code coverage metrics identify untested paths or scenarios, guiding additional test creation to achieve verification completeness.

Verification spans multiple abstraction levels:

- **Behavioral models** for early functional and architectural validation.
- **RTL models** for cycle-accurate functional and interface verification using testbenches, assertions, and coverage metrics.
- **Gate-level models** for pre-layout and post-layout verification, including timing, power, and functional equivalence checks.

Verification frameworks include:

- **UVM (Universal Verification Methodology):** Standardized SystemVerilog-based methodology for building reusable and modular verification environments.
- **OVM (Open Verification Methodology):** Predecessor to UVM, also supports modular and reusable verification components.
- **Cocotb:** Python-based verification environment that interacts with RTL simulators.

Formal Verification

Formal verification is a method that proves the correctness of a hardware design mathematically, rather than relying solely on simulation. It detects design errors which may not appear in simulation that verifies critical properties like protocol compliance, timing constraints, correct data flow and control logic.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Design for Test (DFT)

DFT insertion is performed after synthesis and prior to physical design. During this stage, hardware structures such as scan chains, compression logic, and MBIST/LBIST modules, JTAG macros, and boundary scan cells are added to the gate-level netlist to enable efficient testing of the manufactured chip. Following DFT insertion, the design undergoes DFT verification, which includes:

- **Scan chain:** Ensures proper connectivity of scan paths.
- **Automatic test pattern generation (ATPG) and fault simulation:** Generates and validates test patterns.
- **Coverage analysis:** Assesses the quality and completeness of testability.
- **Timing and area validation:** Confirms that added DFT logic does not violate constraints.
- **MBIST/LBIST verification:** Tests memory blocks and logic structures integrated with self-test capabilities.
- **JTAG Macro:** A standard interface (Test Access Port) used to control DFT features, including scan chains, compression logic, and memory self-tests.
- **Boundary scan:** Ensures that boundary scan cells are correctly inserted between each I/O pin and internal logic, enabling observation and control of chip inputs and outputs for board-level testing.
- **Scan-chain compression:** Ensures compression logic works correctly to reduce test data and time, with an optional smart scan to minimize pin usage.

Physical Verification

Logic equivalence checking (LEC): LEC is a formal verification technique used in the physical verification stage to ensure that the physical layout of a design after post-synthesis, DFT insertion or post-layout netlist is functionally equivalent to its logical schematic or RTL description. This step is crucial to guarantee that no unintended changes or errors were introduced during synthesis, placement, or routing. It ensures that the netlist from the physical layout matches the logical behavior of the original RTL or schematic. It detects netlist mismatches, missing or extra cells, and connectivity errors. LEC focuses solely on logical equivalence, independent of timing, and can automatically verify large designs.

Other methods of physical verification such as DRC, ERC, DFM, and timing and power analysis are described in the backend design flow section.

Timing Verification

Timing verification ensures that an ASIC design meets all specified timing requirements. Primary approaches for timing verifications are dynamic simulation and STA:

Dynamic simulation: Dynamic simulation is performed at the transistor or gate level and captures the effects of gate delays and parasitic on the design's functionality and performance, supporting a wide range of design styles.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Static Timing Analysis (STA): STA is a critical post-synthesis and back-end verification technique that checks whether a design meets its timing constraints, such as setup and hold requirements. STA analyzes the design timing paths to verify that signals can propagate between registers within one clock cycle without violating timing constraints. It is used after synthesis and during post-placement and routing to ensure timing correctness. The key functionalities of STA include:

- Calculating propagation delays along critical paths, including slacks.
- Detecting setup and hold time violations.
- Evaluating clock skew and jitter effects
- Checking multicycle and feasible paths

Power Analysis

Power and energy have become critical considerations in digital hardware design. Power is regarded as equally important as silicon area and performance. Most practical designs consider both delay and energy, with power dissipation arising from two main sources:

Dynamic power is caused by switching activity when transistors are turned on and off during change in signals. Dynamic power consumption increases with increasing frequency.

Static power is caused by leakage currents that pass-through transistors even when they are idle and become more significant with smaller technology nodes.

In ASIC design, reducing switching activity is a key approach to lower dynamic power consumption. Techniques include:

- **Clock gating:** Disables the clock in inactive blocks, directly reducing dynamic power.
- **Glitch reduction:** Minimizes unnecessary transitions caused by unbalanced path delay.
- **Demultiplexing / parallelization**: Reduces extra transitions by avoiding excessive hardware multiplexing, though it may increase area.
- **Multi-voltage domains:** Various sections of the chip operate at different voltages based on their performance requirements.

1.1.4. Physical Design (Back-End)

Physical design, also referred to as the back-end process, is the stage where the actual layout of the ASIC is generated. During this phase, the synthesized gate-level netlist is transformed into the final GDSII file for fabrication and undergoes signoff checks to ensure the design meets all specifications. The back-end ASIC development flow is further sub-divided into the following sections.

Floorplanning

Floorplanning establishes initial high level chip layout by dividing the die into physical partitions, shaping them to meet area and data/control flow requirements, and roughly assigning pins and ports for later placement and routing. It:

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

- Defines core area, I/O placement and space for buffers.
- Allocates space for macros, memories, and standard cells.
- Plans clock tree and power distribution.

Placement

Placement positions standard cells, macros, and IP blocks within the floorplan to minimize wire length, reduce congestion, and optimize timing and power. No actual routing is done, but global routing estimates wire length and congestion. Modern placement engines can use switching activity (SAIF/VCD) information to optimize placement for lower dynamic power.

Power planning

Power planning involves creating power grids and incorporating decoupling capacitors to maintain stable voltage during switching. Effective power distribution is essential to prevent IR drops and electromigration (gradual damage to wires from current flow).

Clock tree synthesis (CTS)

Clock Tree Synthesis distributes the clock across the chip with minimal skew and latency using buffers. Its goals are optimal clock timing and reduced skew, and since clocks toggle frequently, the clock tree can in some designs account for over 75% of dynamic power. Its main task is to generate clock buffers and clock routing, and balance clock arrival times at flip-flops.

Routing

Routing connects all cells and macros according to the netlist across multiple metal layers using detailed routing algorithms. Global routing defines rough paths, while detailed routing assigns precise metal tracks, optimizing timing, signal integrity, and minimal DRC violations. Multiple iterations ensure efficient, legal routes with minimal detours.

Physical layout verification

After completing the layout, tools verify that the design is ready for fabrication. While logic verification ensures correct functionality, physical verification ensures layout is correct. Physical verification involves checking design for manufacturability, reliability, and electrical issues. Such checks include:

- **Design rule check (DRC)** verifies the layout against rules set by chip foundries to ensure it is manufacturable, such as checking for minimum allowed spacing between metal layers.
- **Layout versus schematic (LVS)** verifies the design's functionality by generating a netlist from the layout and comparing it with the schematic netlist.
- **Electrical rule check (ERC)** ensures proper power/ground connections and that signal transition time (slew), capacitive loads, and fanouts are within limits.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

- **Design-for-manufacturability (DFM):** Evaluates the layout for process variations, including dummy metal fills, density rules, and lithography constraints to improve manufacturability and reduce defects.
- **Other checks** include electromigration, ESD, antenna violations, shorts, opens, floating nets, and pattern match violations.

Parasitic-resistance-capacitance (RC) extraction and timing verification

RC extraction converts the layout into a resistance-capacitance network for post-layout analysis. STA uses the extracted RC networks to verify that the chip meets its timing, frequency, and power-performance goals, ensuring all setup and hold constraints are satisfied.

Signoff / tape-out

Signoff is the final stage of ASIC implementation that finalizes the ASIC design for manufacturing, occurring after all physical and timing verifications are complete. This step, also called design tape-out, ensures the design is ready for fabrication. Its tasks include:

- Complete physical and timing verification.
- Generate the final GDSII files for the foundry.
- Confirm that the design meets all layout, electrical, and timing rules.

The finalized GDSII layout is handed over to the chip foundry for fabrication, marking the end of the design phase.

1.2. Analog/Mixed Signal/RF

In this subsection, **analog, analog mixed signal (AMS), and RF design approaches** are considered. These three classes of circuits are grouped together because the associated electrical signals are generally continuous functions of time, or of time and space, reflecting the continuous nature of physical phenomena in the real world.

As an illustrative example, consider an IoT-based temperature control system comprising a temperature sensor, a data logger, a wireless communication module, and a power stage driving a fan. The sensor output typically requires signal conditioning to ensure adequate signal-to-noise ratio for further processing. The power stage amplifies the signal to deliver sufficient power to the fan motor. Temperature control may be implemented using analog strategies such as ON–OFF or PID control. These functional blocks are inherently analog and are designed using classical analog design principles. In contrast, the data logger operates in the digital domain. Therefore, the sensor signal, originally a continuous voltage, must be digitized into a coded representation. This transition between analog and digital domains is addressed by **mixed signal design**, which focuses on interface integrity and system-level modeling across both domains. Finally, the wireless

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

communication module operates at high frequencies, where signal transmission relies on electromagnetic wave propagation. Due to the distinct analysis and synthesis techniques required at these frequencies, such circuits are designed using **RF design methodologies**.

The figure below illustrates a generic **analog/mixed signal/RF design flow**. Each stage of the flow corresponds to an operation performed using one or more dedicated tools, while the arrows indicate the sequence of these operations. As shown, the primary inputs are **(1) the design specification** and **(2) PDK data**, which are specific to the technology selected by the designer.

The **sign-off** stage represents the final verification of the design, ensuring compliance with both the functional specifications and the fabrication requirements of the target foundry. The output of the flow is a fully verified integrated circuit layout, delivered in the standard **GDSII** format.

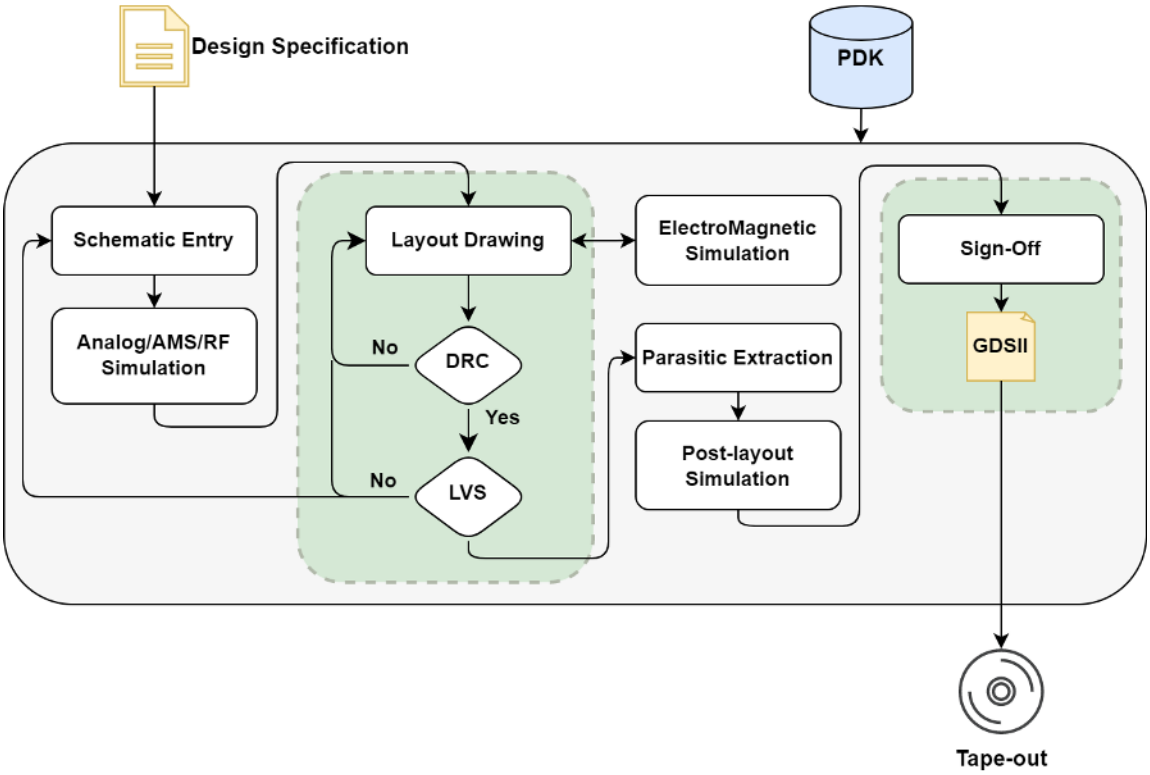

Figure 3. analog/mixed signal/RF design flow.

The following sections describe the **analog, mixed signal, and RF** approaches in detail.

1.2.1. Analog design

Analog design is typically performed at the lowest levels of hierarchy—device and circuit levels—and operates in the continuous domain of physical quantities such as voltage, current, and time. It exploits the continuous behavior of devices to provide gain or implement complex functions in a

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

signal-processing chain, whereas in the digital domain transistors are usually treated as switches. Because analog circuits operate over the full range of available voltages and currents, they are sensitive to noise and variations in temperature, supply voltage, and global and local process fluctuations. Although analog circuits use orders of magnitude fewer components than their digital counterparts, the design process is complex for these reasons and because of cross-coupled dependencies between design steps that affect the final figures of merit. The process is iterative and highly customized, as designs are not portable across processes and scale poorly.

A typical analog flow includes the following steps:

Specification and architecture definition

Usually, an analog block is part of a larger system (as in the IoT example), which imposes external constraints such as input/output signal levels, noise conditions, power-supply requirements, and environmental conditions. From these external constraints, the designer can derive a detailed specification in the form of a set of parameters such as gain, bandwidth (BW) or gain-bandwidth product (GBW), noise, linearity, headroom, PSRR/CMRR, offset, power consumption, area, startup behavior, and reliability targets. This specification serves as the starting point for many analog design methodologies. Similarly to the digital domain, different architectures implementing the same functional block provide the designer with degrees of freedom to make trade-offs and optimize circuit performance and/or cost. Since analog IC design largely consists of sizing circuit components, this process can be carried out using the following methods: (1) calculation of initial values based on analytical equations, followed by iterative, simulator-supported optimization; and (2) systematic device-sizing methodologies such as the g_m/ID approach.

Schematic entry

A schematic capture tool has two major roles: (1) providing a visual representation and documentation of the circuit and its parameters using schematic diagrams and standardized symbols, and (2) translating the schematic diagram into a simulator-readable netlist. Schematic capture tools usually support hierarchical design and allow the user to edit device parameters.

Since the schematic diagram is a method to document the design, additional information like high-current paths and guard rings for device isolation should be clearly documented on the schematic to drive a correct layout development.

Simulation

The main objective of circuit simulation is to ensure correct circuit functionality while accounting for variations in external operating conditions (such as supply voltage and temperature) and imperfections related to the fabrication process (process variations). The specific functionalities to

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

be verified are circuit-dependent, but in general, simulations are set up using a so-called *testbench*—a circuit that emulates a laboratory environment in which different types of instruments can be connected to the circuit under test for evaluation.

Typical analyses include DC (bias) analysis, parameter sweeps, transient analysis, AC analysis, stability analysis (loop gain and phase margin), noise analysis (input-referred and output-referred), transient performance (slew rate and settling time), and PSRR/CMRR evaluation. For periodically driven circuits, additional analyses such as PSS, PAC, PNoise, or harmonic balance are used.

The designer can create as many testbenches as needed and usually starts by assuming nominal operating conditions (temperature and supply voltage) and an ideal fabrication process. It is unlikely that a circuit will meet all specifications after the initial sizing; therefore, the designer's knowledge and experience are required to determine which parameters should be modified or recalculated. Several tools, such as sensitivity analysis, help identify critical parameters that significantly affect circuit performance.

For certain classes of circuits or circuit blocks, local variation simulations combined with dedicated layout techniques are essential to mitigate mismatch effects. Once the circuit meets specifications under typical conditions, the next step is corner simulation, which involves evaluating circuit performance under worst-case conditions. Because PDKs provide information about process variations, yield can also be estimated using statistical circuit simulations based on Monte Carlo methods.

Layout implementation

The implementation of sized circuit elements is carried out in the layout. If mismatch-reduction techniques were considered during simulation, they should be reflected in the layout as well. Other techniques, such as device isolation and noise reduction through the use of guard rings and/or tap rings, should also be documented on the schematic. Electromigration rules must be followed for high-current paths.

Physical verification

Physical verification involves execution of design rule checks and layout versus schematic verification using dedicated engines and rule decks in order to verify the correctness of the implementation and the consistency of the design.

Parasitic elements extraction and post layout simulation.

The metal interconnects in the back end of line (BEOL) introduce parasitic series resistance and capacitance, which must be taken into account for high-speed and high-power circuits. Since the geometry of these interconnects is defined during the layout step and is layout-dependent, the

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

resistances and capacitances are derived from the layout, resulting in a so-called post-layout circuit netlist. This netlist can be simulated using the same testbenches employed during the schematic-level simulation. The extraction of parasitic resistance and capacitance from the layout provides a more realistic estimation of the circuit's figures of merit usually degrading performance what can result in additional circuit optimization round.

Sign-off and documentation

Final examination in the form of a tape out checklist: specification vs. measured in simulation table. It should include physical verification reports: DRC and LVS and also documentation of the design.

1.2.2. Mixed-signal design

Mixed-signal design spans both continuous-time analog and discrete-time/level digital domains, integrating blocks such as ADCs, DACs, PLLs, sensors, and digital control on a single project. It must manage device-level nonidealities (individual electronic noise, local technology mismatching, global PVT variability) while meeting system-level interface requirements (sampling theory, quantization noise, amplitude linearity, dynamic range, and timing constrains). Unlike purely digital designs — where transistors are treated largely as logic switches — mixed-signal circuits must operate across the full voltage/current range and honor strict digital timing, clock-domain crossings, and jitter budgets. Coupled effects like substrate coupling, supply/ground bounce, and electromagnetic interference require careful floor planning, physical and electrical isolation, and power-integrity strategies. Verification combines transistor-level simulation with behavioral modeling and AMS co-simulation to cover long time scales and complex interactions; calibration, trimming, and DFT (e.g., BIST for converters) are often integral to achieving yield. The result is an iterative, co-design process with limited portability across technologies, where architectural partitioning and cross-domain trade-offs strongly determine the final figures of merit. Furthermore, the resulting mixed-signal IP blocks usually need to be integrated in large digital SoCs, thus their design sign-off must also include the generation of the proper physical, timing and power models required by the standard digital design flow.

A general top-down design flow in the case of a mixed signal design is explained in the following sections.

Functional requirements and architecture partitioning

The first step involves defining the functional performance targets (e.g. sampling rate, amplitude dynamic range, jitter time, power consumption, circuit area) according to the overall requirements. According to these functional requirements, a particular mixed-signal architecture is chosen, and its internal functions are split between analog and digital domains.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Behavioral modeling and block specification

To validate the mixed-signal architecture proposal, overall behavioral simulations are executed employing functional HDL models for each block (e.g. filters, samplers, quantizers, oscillators, counters) with two goals in mind. First, evaluate the maximum performance achievable with the selected architecture, considering all its analog and digital blocks to be ideal. Second, find the minimum functional specifications (e.g. SNR, jitter) of each individual constitutive block to keep the overall architecture performance within the initial functional requirements. Thanks to the simplified functional HDL models, the computation time can be reduced. The selected mixed-signal architecture may need to be discarded (i.e. return to step 1) either because its absolute ideal performance cannot meet the minimum initial requirements or because the specifications needed in at least one of the constitutive blocks will not be feasible when being implemented at circuit level.

Block-level circuit optimization

Here, each of the analog and digital building blocks of the mixed-signal architecture are designed. Starting from the individual specifications found in the previous step, the circuit topology of each block is selected and sized following the analog schematic or digital RTL design flows, depending on the nature of each block. These standard design flows have already been explained in the previous sections of this document. Since the complexity of each building block should be relatively low, this part of the local design is usually performed by automated optimization, where the intrinsic constraints come directly from the block specifications while the cost function can be focused on minimizing power consumption, circuit area or PVT sensitivity, among others. Another result from this step of the design flow is a refinement of the block HDL models, which can incorporate now some non-idealities associated with their circuit implementations (e.g. noise PSD, non-linear transfer functions, clock couplings, timing delays, dynamic power profiles).

AMS integration

Once all the constitutive blocks have been already designed at schematic or RTL levels, it is time to validate the full mixed-signal architecture at circuit level. For this purpose, analog/digital co-simulations are carried out with testbenches dedicated for each application scenario. At IP level, these simulations are typically performed following analog-on-top schemes. The measured metrics may not be selected exclusively in terms of performance but also related with initialization, calibration, trimming and any other auxiliary mode of operation. Thanks to the refined HDL block models generated in step 3, the architectural simulations can be executed at different abstraction levels for each building block. Beside the benefits of speeding up simulations, this possibility is usually exploited to easily identify the block circuit causing any misfunction at the architecture level.

AMS floorplanning

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Before attempting the physical design of the mixed-signal circuits, it is a good practice to invest some time in deciding the overall floorplan. This step involves choosing the most convenient placement of the analog and digital blocks according to the architecture interconnectivity to avoid crosstalk at the later routing stage, especially for sensitive analog references. Concerning supplies, it is also important to apply proper partitioning of the supply domains based on the voltage nominal values, continuous/discrete-time operation of the loading circuits and power requirements. In particular, the ground distribution scheme plays a key role in minimizing injections from the substrate in the returning paths. Finally, clock distribution must be carefully checked when mixing continuous and discrete time blocks.

AMS layout and verification

For the layout implementation of the mixed-signal architecture circuits, the specific physical design guidelines will be followed for each building block, depending on its analog or digital nature. For the analog case, device technology matching and signal decoupling are the two main driving vectors of full-custom layout design, with the corresponding penalties in terms of area. On the contrary, density and timing are common goals when executing the automated P&R for digital blocks. Following the previous sections, special care must be taken to provide physical guarding and shielding within both signal and supply domains. In terms of methodology, the standard physical verification checks apply here (i.e. DRC, LVS and ERC).

Post-layout simulation

As in the standard design methodology of separated analog and digital circuits, parasitic extraction (PEX) is a key step to complete the physical verification of any mixed-signal layout. Depending on the frequency range and dynamic range of the target application, the analog parts may require a complete RLC PEX or a simplified PEX can be accurate enough. As per the digital constitutive blocks, the PEX results are usually back-annotated into the architecture for the identification of new critical paths. At the end, the same testbenches and analog-digital co-simulations carried out during the AMS integration step described above need to be redone with the new PEX data to re-verify the overall performance figures.

Test and DFT

Due to the complexity of mixed-signal architectures, it is very common to add some inner testing capabilities to be used after fabrication through the adoption of design for testability (DFT) strategies. Such DFT tools may include but are not limited to adding analog hooks to increase observability or extending the functionality of the digital interface to increase controllability of the analog domain. Standard digital DFT techniques (e.g. scan path, BIST) also apply here.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Digital-on-top IP integration and sign-off

As part of the design sign-off of a mixed-signal IP design, the views required for its usage in a digital SoC are usually generated. Typically, these views describe the mixed-signal IP as a digital macro cell for its smooth insertion into the standard digital design flow (e.g. LEF and Liberty views).

1.2.3. RF design

RF and mmWave integrated-circuit (IC) design focuses on the physical realization, modeling and verification of transceivers and front-end subsystems operating at frequencies ranging from tens to hundreds of gigahertz. Designing at these frequencies introduces challenges not present in conventional analog IC design: circuit performance is strongly influenced by distributed electromagnetic effects, intrinsic device limits (e.g. ft and fmax) and parasitic coupling. On-chip passive structures, including transmission lines, inductors and transformers exhibit finite quality factors and can experience strong mutual coupling, while package and antenna interfaces introduce additional parasitic impedances, losses, and couplings.

The RF IC design flow progresses through iterative stages, beginning with system specification, where functional behavior, frequency bands, performance, power, and area budgets are defined. These requirements are distributed between blocks in the block architecture and budgeting step, guiding the block design and schematic, where circuit topologies, operating points, and internal block-level specifications are established. Pre-layout simulation validates functionality under nominal and corner conditions using both analog and RF-specific analyses. During the layout stage the physical geometry is created, and RLC extraction along with EM simulation capture parasitics and distributed effects for accurate modeling. Physical layout verification ensures design rule compliance, electrical correctness, and reliability. Post-layout simulation confirms targeted performance while accounting extracted and EM-aware models. Calibration and test strategy integrates trimming, monitoring, and test structures, while package and antenna co-design accounts for the introduced parasitics, coupling, and radiation. Finally, sign-off consolidates all verifications to approve the IC for fabrication.

System specification

The first step in RF IC design involves defining the system's functional behavior, including external interactions, overall performance and reliability objectives. At this stage the application context must be clear and the overall architectural concept and module interfaces are established without specifying detailed circuit-level requirements. Key operational scenarios are defined and their relevant use cases may be tested using high-level behavioral or system-level models, enabling the evaluation of architectural implementation, performance trade-offs and functional completeness. The system-level budget for the main resources, such as operating bands, channel plan, link budget,

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

gain, noise, linearity, output power/PAE, phase noise/jitter, EVM, spur masks, isolation, power consumption and silicon area are defined based on the target application constraints.

Block level architecture and budgeting

The high-level system architecture is translated into a concrete block-level architecture having functional blocks with well-defined interfaces, signal flows, and preliminary performance targets. This includes signal levels, port impedances, noise contributions and power-supply limitations. This transition step bridges the gap between system-level requirements and detailed circuit-level implementation. A top-down budget allocation is performed, distributing the system-level budget requirements among individual blocks to ensure overall system-level compliance. Candidate block architectures are evaluated and compared to explore trade-offs between performance, robustness, and implementation cost.

Block level design and schematic

For each functional block, the design focuses on translating the block-level budgets, allocated in the previous step, into internal circuit-level requirements. This involves defining the block topology, biasing strategy, and operating points, as well as specifying internal performance limits such as noise and distortion contributions, phase noise masks, headroom constraints, impedance targets, supply and substrate sensitivity, and PVT robustness.
Once the block-level topology is finalized, the design proceeds to the schematic-level representation, where functional correctness, stability, and preliminary performance are verified through simulation.

Pre-layout simulation

In the design of RF integrated circuits, pre-layout simulation is a crucial step aimed at validating circuit functionality and verifying compliance with the allocated block-level specifications prior to physical implementation. The primary objective of circuit simulation is to assess performance under both external variations, such as supply voltage and temperature changes, and internal non-idealities introduced by the fabrication process, including process variations and device mismatch.

The choice of simulation type and analysis depends on the specific circuit block and its role within the RF system. While RF IC design shares many similarities with general analog design, RF circuits require a set of specialized analyses to accurately capture frequency-dependent and nonlinear effects.

At the initial stage, fundamental analog simulations are performed to ensure correct biasing and baseline functionality. These include DC operating-point analysis to establish bias conditions, parameter sweeps to evaluate sensitivity to design variables, transient analysis to inspect time-domain behavior, and AC analysis to assess small-signal frequency response. Stability analyses are also conducted at this stage to ensure reliable operation across the intended operating range. In

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

addition, power supply rejection ratio (PSRR) and common-mode rejection ratio (CMRR) analyses are performed to evaluate immunity to supply and common-mode disturbances.

For RF-specific performance evaluation, S-parameter simulations are extensively used to characterize small-signal gain, bandwidth, and input/output matching across the operating frequency bands. Additionally, other analyses such as stability factor evaluation (Rollett's K and μ factors), noise and gain circles, and matching network optimization are commonly employed to guide the design of RF blocks.

To accurately capture nonlinear and large-signal behavior, more advanced RF simulations are required. Harmonic balance (HB), periodic steady-state (PSS) and related analyses such as periodic AC (PAC), periodic noise (PNoise), quasi-periodic steady-state (QPSS), and source- or load-pull simulations are used to identify distortion mechanisms, evaluate steady-state operating conditions and optimize performance under large-signal excitation. These analyses enable the extraction of key RF figures of merit, including gain compression, intermodulation distortion, output power, efficiency and phase noise, as well as the sizing and optimization of matching networks for maximum power transfer and minimum noise contribution.

The simulation process in RF IC design begins with initial component values and device dimensions, typically estimated using analytical models or structured sizing methodologies. These estimates are subsequently refined through iterative, simulator-assisted optimization, incorporating RF-specific performance metrics, until all target specifications are met. Sensitivity and corner analyses are employed to identify dominant performance drivers, enabling focused design refinements and efficient convergence toward a robust pre-layout solution.

Layout

The layout stage translates the RF circuit schematic into a physical geometry. In the RF circuit design flow, the layout step represents a critical phase of IC realization since parasitic effects, coupling, and substrate interactions significantly influence high-frequency performance. Consequently, the layout process is inherently iterative: multiple refinements are carried out to prevent performance degradation while simultaneously ensuring manufacturability, reliability, robustness, and compliance with design rule checks, electrical checks and manufacturability assessments.

Place and Route

In RF IC design, device and component placement are guided by the circuit hierarchy and signal flow, with functionally related blocks grouped together to minimize interconnect length, parasitic capacitance, and inductance. Symmetry is rigorously enforced in differential and balanced circuits to reduce mismatch, phase errors, and even-order distortion, ensuring optimal signal fidelity.

Sensitive RF blocks are carefully isolated using guard rings, shielding metals and deep n-well structures, mitigating substrate noise, magnetic coupling, and interference from nearby digital or

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

high-power sections. High-power blocks are placed with particular attention to thermal management, isolation, and current-density distribution to prevent performance degradation and electromigration issues.

RF signal paths are routed to be short, straight, and symmetric, limiting resistive losses, parasitic inductance, and phase imbalance. Critical signal traces are often routed over low-loss substrates, with controlled impedance and minimal crosstalk. Low-impedance return paths are ensured through dedicated ground routes, planes, and rings, reinforced with multiple distributed vias, to reduce ground bounce, noise coupling, and common-mode disturbances.

Placement and routing decisions are iteratively validated using parasitic extraction, electromagnetic simulation of interconnects, and post-layout verification to ensure that layout-dependent effects do not compromise RF performance.

RLC Extraction

In RF circuit design, RLC with coupling extraction translates the layout geometry into an equivalent electrical network composed of resistances, capacitances, and inductances including mutual coupling effects. This extracted RLC network captures parasitic and coupling phenomena introduced by interconnects and passive structures, and it is used (together with electromagnetic simulations of passive structures) for post-layout simulation to assess and verify the impact of layout-dependent effects on circuit performance.

Electromagnetic Simulation (EM)

In RF circuit design, electromagnetic (EM) simulation of passive components, routing and passive structures models the high-frequency electromagnetic behavior of the implemented layout, enabling accurate representation of distributed effects that cannot be captured by lumped-element models.

EM simulations capture critical phenomena such as parasitic coupling, electromagnetic radiation, substrate losses, skin and proximity effects, and frequency-dependent impedance variations. They are used to extract accurate S-parameters or equivalent models for passive components and interconnects, which are then used into post-layout circuit simulations.

This process is essential both for the verification and optimization of component placement and routing, and for the design of custom passive elements (such as inductors, transformers and transmission lines) which are typically drawn and iteratively refined through EM simulation across multiple layout iterations.

Physical layout verification

In RF IC design, physical layout verification is performed during and after the layout process to ensure that the device is ready for fabrication, functionally correct, and robust against manufacturing and operational issues. This verification involves a combination of design rule

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

checks, electrical checks, and manufacturability assessments to guarantee reliability and performance. Such checks include:

Layout Versus Schematic (LVS) verifies that the layout implements the intended circuit functionality by extracting a netlist from the layout and comparing it to the original schematic netlist, detecting connectivity errors, missing or unintended devices.

Design Rule Check (DRC) ensures the compliancy of the layout to the geometric and spacing rules specified by the foundry, such as minimum and maximum metal widths, spacing between layers and enclosure rules to guarantee manufacturability and prevent fabrication defects.

Electrical Rule Check (ERC) confirms correct power and ground connections and detects layout issues that could cause malfunction, reliability problems, or significantly degraded performance.

Design-for-Manufacturability (DFM) evaluates the layout against process variation sensitivities, lithography limitations, metal density requirements and the need for dummy fills to ensure reliable fabrication and reduce defects.

Electrostatic Discharge Vulnerabilities (ESD) checks that the ESD protection structures (diodes, clamps) are correctly connected to all sensitive nets to avoid any damage caused by static discharges.

Reliability Analysis includes checks for electromigration (EM), IR drop, antenna effects, shorts, opens, floating nets and pattern match violations that could compromise functionality and long-term reliability or cause failures during fabrication.

Post-layout simulation

After the layout realization, the circuit simulations are extended to include extracted and electromagnetic (EM) views, and the previously cited simulations (S-parameters, noise, stability, etc.) are repeated to ensure that performance remains within specification after layout effects are included.

Finally, corner simulations are performed to evaluate performance under worst-case conditions of process, supply and temperature, while Monte Carlo simulations account for mismatch effect impact by statistically sampling process variations.

Through an iterative simulation process, the RF IC designer gradually converges toward a design that reliably meets performance targets across all anticipated operating conditions. The combination of simulation and statistical evaluation of schematic, extracted and electromagnetic views ensures robust circuit operation and manufacturability.

Calibration and test strategy

In RF IC design the calibration, control, and test strategy step defines how the IC will be trimmed, monitored, and tested to ensure performance meets specifications. This step includes planning on-

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

chip calibration mechanisms such as trimming circuits to correct I/Q imbalance, LO leakage, DC offsets, and digital predistortion (DPD) controls for power amplifiers. It also involves integrating on-chip detectors and test access points to enable accurate characterization during wafer probing and production testing. Furthermore, on-wafer test structures are usually included for de-embedding strategies such as Thru-Reflect-Line (TRL) or Short-Open-Load-Thru (SOLT).

Package and antenna co-design

This step ensures that circuit and system-level performance targets are met when the integrated circuit is assembled. In this phase, the electrical behavior of the package is modeled together with the antenna, including all relevant elements such as solder bumps, bond wires, vias, transmission lines and antenna feeds. These elements introduce parasitic impedance, loss, and coupling that can significantly affect matching, radiation efficiency, and bandwidth. Full-wave electromagnetic simulations are used to evaluate mutual coupling in antenna arrays, radiation and beam patterns and their sensitivity to package layout and materials. The extracted package and antenna models are then incorporated into the RF system simulation to assess key metrics such as error vector magnitude (EVM) and adjacent channel leakage ratio (ACLR), ensuring that modulation accuracy and spectral compliance are maintained when the package parasitics are accounted.

Sign-off/tape-out

During signoff, a comprehensive verification is performed to ensure that the layout meets all foundry design rules, electrical constraints, timing requirements, and RF-specific performance targets. Key tasks in this stage include the generation of the complete, fully verified GDSII files for the foundry, the preparation of supporting documentation and test plan guidelines.

1.3. Integrated Photonics

PIC design concerns simulation, layout, and performance analysis of integrated circuits capable of processing optical signals in the visible and near- / mid-IR wavelength range (from 400 nm to above 2 um). There are numerous material platforms which enable such functionality, including but not limited to InP, SiN, Si-on-insulator, polymers, etc. Typical functional blocks include passive components (waveguides, splitters, combiners, reflective elements), active components (light sources, amplifiers, and photodetectors), phase modulation components, and polarization management building blocks. More complex components such as lasers, intensity modulators, (de-) multiplexers are either built by designers using these primitive blocks or provided by the foundries / IP vendors. Many of the mentioned functional components essentially enable signal conversion between optical and electrical (both DC and RF) domains, therefore PIC design also deals with design of electric circuits driving and controlling these components. Some steps, methods, and software relevant for analog and RF design (see previous section) are also relevant for PIC design.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

In the described flow, we omit the system-level design stage, and assume that the system specifications are available beforehand as they define the target performance for the PIC-based device.

PIC design steps can be grouped in the following 4 stages: conceptual design, (optional) block-level design, circuit simulation and layout, and physical layout verification. *Conceptual design* stage is an idea evaluation and technology selection stage. *Block-level design* is an optional stage which is normally done by designers involved in platform or IP block development. A typical design flow assumes usage of existing IP blocks (provided by foundry or IP vendor), therefore a typical designer will not be involved in this stage. However, the relevant steps and tools are mentioned in this report for completeness. *Circuit simulation and layout* is the main PIC design stage, which is often an iterative process between the involved steps. Finally, *Physical layout verification* is performed on the completed design and can be run by both the designer and the foundry.

1.3.1. Conceptual design

The steps in conceptual design can be described as follows.

Conceptual schematic design

This step involves creating abstract circuit schematic for the desired functionality. This involves understanding of typic functional blocks and their theoretical behavior in analytical form. The result of this step is conceptual ("powerpoint") circuit diagram, list of component specifications and power budget.

Technology choice

Analysis and evaluation of available technologies, material platforms. What are the platform limitations, customization possibilities? System-level technical requirements and application-related specifications, as well as general platform specifications are used on this step. The result is the selected technology and a list of specific building blocks to be used in the PIC.

Concept evaluation

The concept evaluation step includes schematic detailing, performance estimation and analytic calculations for the selected platform. Application-related specification concerning e.g. power budget, target bandwidth and precision are addressed here. Detailed performance information from the design manual and the PDK is used. Successful evaluation at this step gives green light for the circuit simulation and layout phase, and provides functional specifications for the circuit.

1.3.2. Block-level design

Block-level design is an optional stage which is done during technology or IP block development. It requires more detailed information about the fabrication process than is normally available in the photonic PDKs, therefore a typical PIC design flow doesn't involve these steps and the related tools.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Layer stack design

Layer stack design involves material composition design, layer thicknesses definition and simulation. Involves process capability data. Process data itself is not needed. Combined with epitaxial process design, design intent can be mapped to process recipes. Subsequent inline and offline measurements can correlate these data to functional specifications such as gain and loss.

Custom building block design

This step concerns definition and optimization of building block layout for specific metrics. At this step, a building block is treated as a white-box, i.e. all the geometry of mask layers is known and can be adjusted. To do this, quantified layer stack descriptions and design rules are needed. The result is the optimum mask layout of the building block. Electrical, optical, electro-optical and temperature performance of the building blocks are simulated here.

1.3.3. Circuit simulation and layout

The following describes the steps involved in circuit simulation and layout of PICs. Note that although mentioned sequentially, the actual order of the steps is not linear. Circuit simulations and layout are often repeated iteratively in a loop, because only on the layout stage the precise geometry is determined, and the geometry of components (e.g. waveguide shape) is important for circuit behavior. Design for testing has to be done at every design step to ensure there is a matching validation and verification step in the product test stage (V-model).

Optical / DC circuit simulation

At this step, static circuit performance is analyzed. The electrical and optical simulation are performed independently from each other. Simulations use circuit schematic, performance data of the building blocks (numeric or compact models, e.g. S-parameters). The methods include S-matrix simulation, frequency and time domain simulations. The optical simulations cover parasitic reflections in the circuits. Electrical simulations may include SPICE simulations, as electrical part of the circuit is effectively an analog electric circuit. Optimization of circuit performance against design parameters is done at this step too.

RF circuit simulation / co-simulation

The next step in circuit simulation is dynamic circuit behavior simulation (transient analysis), RF performance simulation, and electro-optical co-simulation. In addition to the methods used in the previous step, more advanced models which take into account both optical and electrical component behavior are used. Similar to the previous step, the design parameters may be optimized here.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Variation simulations

To simulate how robust the circuit is to the fabrication variations, this type of simulation is performed. At this step, the circuit and individual block parameters are fixed, and the simulation determines the performance variation. Critical To Quality (CTQ) parameters are identified here. Circuit simulators often use Monte-Carlo simulations and analysis at this step.

Layout

Layout step includes floor planning, component placement, and routing of electrical and optical interconnects. There are schematic-driven tools, but it is still popular to script the circuit layout manually. Various design rules are taken into account in this step, e.g. component placement, thermal and cross-talk rules, etc. As a result of this step, a mask file is generated.

Design for packaging

Design for packaging groups several activities related to packaging of the designed PIC. The packaging requirements covering layout (e.g. locations and number of IOs) and performance (thermal management, stability, environmental conditions) are taken into account on specifications analysis and layout stages. Package affects RF performance of the system, therefore reiterating RF simulations taking the package into account is necessary. Assembly procedure is designed at this step, which may lead to adding special structures / fiducials enabling compatibility with assembly tool processes. Assembly Design Kits (ADK) are used at this step.

Design for testing

This step guarantees that the die is designed for test against the goals of the experiment or specification of the product, is compatible with a specific measurement setup (both from layout and signals point of view) and contains the structures necessary for probing and alignment processes. The complete test procedures including Design of Experiments (DoF) and data analysis is defined at this step. Placement of additional test circuits and sub-circuits, adding alignment markers, etc. is also done here. For circuit prototypes, it is important to have additional sub-circuits or dedicated test circuits which allow debugging. Design software uses Test Design Kits (TDKs) when performing these tasks.

1.3.4. Physical layout verification

Physical layout verification is performed regularly during the layout steps, and it guarantees that the device is manufacturable in the given process, and functionally correct. Currently, some of these checks are only possible to perform at the foundry side, as this requires replacement of black boxes with white boxes. Note that the term “physical layout verification” is not commonly used yet in the PIC community and this term is used to align with the other design flows.

Mask-level Design Rule Check (DRC)

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

This step involves checking of requirements on mask layers, e.g. spacing, minimum and maximum spacing, overlaps, etc.

Layout versus schematic (LVS), Electrical Rule Check (ERC) and Optical Rule Check (ORC)

LSV extracts circuit schematic from the mask layout, and checks its equivalence with the original (target) schematic. ERC and ORC check correctness of the optical and electrical connections between the building blocks, interconnects, pads, and IOs.

Post-layout performance analysis

Analysis of optical / electrical cross-talk, temperature effects, etc. based on the final circuit layout. In PIC design, this step is currently quite immature, but will definitely play a larger role in the future.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

2. Open-Source EDA Tools

This section describes the open-source tools used in each of the flows. First, considerations about selecting and integrating open-source projects are discussed. Second, descriptions of the tools are given in tables. Third, feature lists and functionalities of the tools are compared in another set of tables.

2.1. Open-Source-Related Considerations

Open-source tools vary in the way they are owned, managed, and developed. Some of the factors related to it are described below.

First and most obvious consideration is the project license. They range from very permissive (MIT, BSD) to licenses which protect open-source nature of the code including its derivatives (GPL). The license conditions may also propagate onto the output created with the toll, such as IPs or PDKs. Therefore, the license should be carefully studied before including a tool in a concrete workflow.

Open-source projects differ in the provided support level, documentation quality, pace of bug fixing and introduction of new features. This depends on the maturity of the project, as well as size and availability of the maintenance team. It is possible to evaluate these characteristics by checking the code repository metrics such as number of project stars, forks, maintainers, number of long-pending issues, and frequency of releases.

Finally, there is a significant difference between open-source projects that are "published source" versus "collaborative open source". In the first case, anyone can clone, modify, repackage, etc. the code, but the publishers do not actively encourage (or handle) feedback and feature requests. In collaborative open-source projects, the maintainers actively encourage feedback through, for example, a public issue tracker and GitHub pull requests (to accept code fixes from external parties). For some use cases, relying on "published source" tools could be considered a risk because these types of projects run a higher risk of becoming abandoned by the owners, or at the very least tend to lag behind to latest innovations and developments.

2.2. Digital

As semiconductor technology advances, design challenges such as power optimization, efficiency, cost, and size have become increasingly significant. To overcome such challenges, specialized EDA tools are applied at each stage of the design flow. Over the past few decades, a few major EDA companies have dominated the market, but due to the high cost of licensing, open-source alternatives are increasingly gaining popularity. Various open-source EDA tools and frameworks are

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

listed below that cover the complete digital design flow, from system-level architectural exploration to tape-out. The tools are ordered from higher levels of abstraction in the ASIC design flow, starting with system-level design and multi-feature frameworks and down to tape-out or more specialized tools with less features. Some tools have multiple functionalities that can be used in system level architectural design exploration to RTL verification, hence if the tools is already defined in earlier section such as at system level, then it's not further included and defined in later sections such as at RTL verification.

2.2.1. Tool descriptions

Tool descriptions are summarized in Table 2.1. Tools that are closely related or share similar features are grouped within the same section. In addition to EDA tools, the table also includes relevant frameworks and supporting toolchains.

Table 2.1. Digital Design Tool Descriptions

Tools Description	
System-Level EDA	
System Design & Architecture Exploration: Simulator, Virtual Prototyping Explore architecture choices, performance, and memory hierarchy before RTL is written.	
FireSim	FPGA-accelerated full-system hardware simulation platform, enabling RTL validation and co-simulation with cycle-accurate models, from single-board setups to large-scale cloud FPGA deployments.
Gem5	Full-system, modular, multi-ISA computer architecture simulator for design-space exploration. This platform is intended to explore computer system architecture, including both system-level designs and processor microarchitecture. It is primarily used to assess new hardware designs, test system software modifications, and evaluate optimizations applied at compile-time or run-time.
GVSoC	An open-source simulator targeting the PULP RISC-V architectures that can simulate complex full-platforms, including multicore, multi-memory levels (i.e., on- and off-chip), multi I/O peripherals, and accelerators, laying the bases of a real DSE for IoT embedded systems.
QEMU	Machine & user space emulator and virtualizer supporting multiple CPU architectures, capable of full system or user-mode emulation entirely in software.
Renode	Generic and open-source machine emulator that aims to simulate platforms consisting of processor cores and peripherals for multi-node embedded networks, enabling a scalable workflow for building reliable, tested, and secure IoT systems.
Structural Simulation Toolkit (SST)	Explore highly concurrent systems where the ISA, microarchitecture, and memory interact with the programming model and communications system.
VPSim	Virtual Prototyping Simulator is a design environment that enables rapid design space exploration, virtual prototyping, and high-level validation, during the design phase of complex digital systems. Its simulation method separates processors and memory hierarchy for efficient design space exploration.
MuchiSim	Simulator framework for analysis of PPA and cost of multi-node multi-chiplet tile-based manycore designs.
noxim	Network on Chip Simulator.
Ramulator	Modular, and extensible cycle-accurate DRAM simulator.
SimEng	HPC Simulation Engine. Fast, easily modifiable, cycle-accurate processor simulator framework

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Spike	RISC-V ISA Simulator.
SystemC (TLM)	Transaction Level Modeling (TLM) is a standard interface for SystemC, a framework for virtual prototyping that enables modeling and simulation for architectural analysis, SW development, performance analysis, and HW verification.
Power Estimation	
McPAT	Architectural integrated timing, area and power modeling framework for multicore and core processors.
IP / HDL Design, Generation and Integration	
Switchboard	Communication and HW/SW Co-simulation framework for communication between diverse hardware models, including RTL simulations, FPGA-based RTL, and fast software models.
FuseSoC + Edalize	FuseSoC: open-source package manager and build tool for reusable IP cores and SoCs. Edalize: Backend abstraction library for EDA tools, used with FuseSoC to generate simulator/flow scripts.
Kactus2	An open-source IP-XACT-based design tool for embedded systems and SoCs. It provides a metadata-driven environment for sketching, packaging, integrating, and generating both hardware and software for system designs. It supports creating and managing IP libraries in the IEEE 1685/IP-XACT standard format, helps in assembling design hierarchies. The tool facilitates reuse of IPs, IP integration, and maintaining consistency in design metadata.
MESSY	An open-source framework that integrates functional RISC-V simulation (achieved with GVSoC) with SystemC-AMS (used to model extra-functional aspects, in detail, power storage and distribution). Allows to perform a DSE that is dependent on the mutual impact between functional and extra-functional aspects like power consumption.
Front-End: Logic Design (HDL) & Verification	
HDL and HLS Framework, Domain - specific IP Generator (HW-SW Co-design)	
CIRCT	Circuit Intermediate Representations (IR) Compilers and Tools, an open-source HW compiler infrastructure and HW generation HLS tools built using MLIR.
Dynamatic	HLS compiler that produces synchronous dynamically scheduled circuits from C/C++ code with dynamic scheduling. It generates synthesizable RTL with better and similar performance to commercial HLS tools in some specific areas. Its fully automated compilation flow is based on LLVM.
PandA-bambu	High-level synthesis (HLS) Framework that enables research on HW/SW co-design. It includes methodologies supporting research on HLS of hardware accelerators, parallelism extraction for embedded systems, hardware software partitioning, and mapping.
PipelineC	Open-source C-like HDL with HLS-like automatic pipelining as a language construct/compiler feature. It can be partially compiled by gcc/llvm for basic functional verification, simulation and compiler produces synthesizable VHDL.
XLS	Accelerated HW synthesis HLS framework that produce synthesizable designs (Verilog/SV) that's aiming to be SDK enabling rapid development of hardware IP. It is still experimental. It supports both optionally pipelined functions with simple wire-based interfaces and concurrent processes.
OpenASIP	Application Specific Integrated Processor (ASIP) toolset for designing and programming compiler-programmable accelerators. Initially based on Transport Triggered Architecture (TTA), it also supports RISC-V co-design. The toolset provides a retargetable flow from high-level programs to synthesizable RTL (VHDL/Verilog) and parallel binaries, with customization of register files, function units, operations, and interconnects. It supports simulation, architectural exploration, and RTL generation, enabling optimized, application-specific processors, and the cost metrics typically involve area, performance, and/or power consumption.
Amaranth (nMigen)	Toolchain for developing hardware based on synchronous digital logic using the Python along with SoC toolkit and more, with an objective to simplify hardware design complicity with reusable components. Amaranth toolchain consists of Amaranth HDL, standard library, simulator, build system. Verilog, SV and VHDL can be integrated into its flow or Amaranth code into Verilog based design flow.
Chipyard	Open-source agile integrated SoC design framework that supports hardware development for RISC-V-based systems. It consists of collections of tools and libraries including processors, accelerators, memory subsystems, and peripherals, and provides tooling for simulation.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

	Chipyard unifies hardware generation (using Chisel and FIRRTL), simulation (software and hardware-accelerated), and system-level experimentation, enabling designers to build complete SoCs with reusable IP components.
Chisel	Open-source Scala based new HDL is used to design advanced reusable, parameterizable digital electronics and circuits at RTL level for ASIC and FPGA systems. It can generate high speed C++ based cycle accurate software simulator or low-level synthesizable Verilog design for synthesis, place and route.
Py2HWSW	Python framework for HW/SW co-design, project management, flow automation, and Verilog generation. It generates lint-friendly and portable Verilog code that can be ported seamlessly between FPGA and ASIC flows.
IMPRESS	IMPRESS (IMplementation of Partial REconfigurable SystemS) is an open-source reconfiguration tool for implementing multi-grain reconfigurable systems in Xilinx Series 7 FPGAs and Zynq-7000 SoC.
bsg_fakeram	Generate black-boxed SRAMs
Lake	Framework that generates synthesizable memory modules from high level specification and widely available memory macros. It consists of generalized hardware modules aimed at memory controller designs.
OpenRAM	Python-based SRAM generator for ASICs that creates layouts, netlists, timing and power models, placement and routing information, and all necessary views for integrating SRAM into ASIC designs.
RgGen	Register Generator. Automatically generate source code for control and status register (CSR) such as SystemVerilog RTL, UVM register model, C header file, wiki doc and human readable format register map specification. Supports standard bus protocols.
RTL Compilation / Simulation (Optionally Synthesis)	
essent	(Essential Signal Simulation Enabled by Netlist Transformation) High-performance RTL(FIRRTL) simulator (cycle-accurate RTL simulators), incorporates various optimizations for improving performance which can be activated or deactivated with command line option.
GHDL	VHDL analyzer (analyze and elaborate), compiler, simulator, and experimental synthesizer. Full support for 1987,1993, 2002 version of IEEE 1076 VHDL standard, and partial support for 2008 and 2019 revisions, and partial support of PSL (Property Specification Language). Can handle large design, comprehensive tool for designing, testing, simulation, verification and partially synthesizing the VHDL code. Supports Co-simulation typically with C/C++/SystemC supported via Verilog Procedural Interface (VPI) and VHDL Programming Interface (VHPI). Can be integrated seamlessly with other popular open-source EDA tools for streamlining simulation and synthesis workflows. Support OSVVM, UVVM and VUnit testing and verification framework.
Icarus Verilog	Verilog compiler and simulator, intended to compile all Verilog HDL as per IEEE 1364 standard but not there yet. Also supports SystemVerilog partially and some of its extensions.
NVC	VHDL compiler and simulator. Supports 1993, 2000, 2002 revisions of VHDL and almost all of VHDL-2008 with exception of PSL and experimental support for Verilog and VHDL-2019 are being developed. Supports OSVVM, UVVM, VUnit and cocotb verification and testing frameworks, and subset of VHPI.
Surelog	SystemVerilog 2017 Pre-processor, Parser, elaborator, UHDM (Universal Hardware Data Model) compiler. Provides IEEE Design/TB C/C++ VPI and Python AST API. Aiming for a complete SystemVerilog 2017 frontend: a preprocessor, a parser, elaborator for both design and testbench, supporting all opensource cores. Can be used by Linter, Simulator, Synthesis tools and Formal tools. It can either be developed as plugins or frontend as an intermediate step for compilation flows.
Synlig	SystemVerilog synthesis tool which uses Surelog as a SystemVerilog 2017 frontend (preprocessor, parser, elaborator) with Yosys as a synthesis framework. Also available as Yosys plugin.
Verilator	Verilog/SystemVerilog simulator, compiler and lint system. Its package converts Verilog and SV HDL design to C++/SystemC model after compilation which can be executed.th. Optionally insert assertion checks, and coverage analysis points. With recent developments, it can also parse UVM code and compile some UVM testbenches. Usually for compilation and faster optimization of code which is then wrapped into C++/SystemC or for high-speed simulation of design. Supports

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

	cocotb, C++/SystemC, UVM (Universal Verification Methodology) testbench and verification framework.
Waveform Viewer	
fstdumper	Verilog VPI module to dump FST (Fast Signal Trace) databases. Aiming to provide unified VPI modules that work across most simulators with strong VPI support to dump simulation waveform in .fst format which can be viewed using GTKWave.
GTKWave	Full featured GTK+ based waveform viewer for large system or designs that read FST, GHW, LXT, LXT2, VZT and standard Verilog VCD/EVCD files. Primarily used for debugging Verilog or VHDL simulation models, specially designed for post simulation analysis of dumpfiles rather than real time interaction during simulation. Allows visualization of both analog and digital signals.
simview	Text-based SystemVerilog design browser and waveform viewer. Run over SSH connection, intuitive control of UI. Generally, for debugging and analyzing SystemVerilog HW design and limited support for non-synthesizable testbench code. Features with full design parsing & elaboration powered by slang, trace signal drivers/loads, support VCD & FST (format written by Verilator) waveform, automatic or manual matching of wave file hierarchy to design hierarchy and send signals from source code to wave & vice-versa.
Sooty	Graphical command-line tool for visualizing vdc waveforms from hardware simulations, as an alternative to GTKWave. Lightweight, easy to use, leverage modern terminal features to provide clean graphical display with customizable display style.
Surfer	Extensible and snappy waveform viewer with web interface supporting VCD, FST and GHW as well as memory transaction format FTR.
Linter	
PySlint	Python (pyslang) based SystemVerilog linter for testbenches, UVM, DPI, functional coverage, RTLI and SVA.
SVA Lint	Linter for SystemVerilog Assertions (SVA). Minimalist linter designed to style and consistency rules for SystemVerilog code which is based on BYOL (Build Your Own Linter) that allows users to define custom linting rules for specific needs. It is flexible and extensible.
svlint	SystemVerilog linter compliant with IEEE1800-2017, based on sv-parser that can be integrated into text editor.
TerosHDL	Toolbox for HDL(ASIC/FPGA) developers, offering VHDL and Verilog/SystemVerilog IDE support, including error and style linting, code formatting, code snippet & grammar checking, templates generation, dependency, automatic documentation, state-machine visualization (design & viewing), and hierarchy, dependency and schematic viewing. Also, supports tools such as GHDL, Verilator, Icarus Verilog, Yosys, VUnit, and cocotb. Example Support VSG as VHDL style linter, and Verible ad Verilog/SV style linters.
Verible	SystemVerilog developer tools, including parser, style-linter, formatter and language server. Easy to integrate with GitHub.
Formal Verification	
ABC	A System for sequential logic synthesis and formal verification. It can also optimize, map and retime industrial gate-level designs with 100K gates.
AutoCC	Frontend for SymbiYosys (SBY) to automatically discover covert channels in time-shared hardware.
AutoSVA	Tool that automatically generates formal property verification testbenches and SVA properties for unit-level RTL verification.
cvc5	Formal verification tool, included in Yosys or SymbiYosys. It is an automatic theorem prover for Satisfiability Modulo Theories (SMT) problem and also provides Syntax-Guided Synthesis (SyGuS). It is 5th generation in Cooperating Validity Checker (CVC) family (CVC, CVC Lite, CVC3, CVC4).
eqy	Equivalence checking with Yosys, a tool for formal verifications to ensure functionality of design is not altered by synthesis tool.
mcy	Mutation coverage with Yosys to improve test coverage, a coverage metric for testbenches. Filters false positives with formal equivalence checks, can be interfaced with other formal tools.
SBY	Front-end for Yosys-based formal hardware verification flows. It provides flows for bounded verification of safety properties (assertions), unbounded verification of safety properties, generation of testbench from cover statements, and verification of liveness of properties.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Verification & Testing Framework	
cocotb	Testbench environment for verifying VHDL and SystemVerilog RTL using Python. Fast, easy and reusable.
OSVVM	Open-source VHDL verification methodology, VHDL verification framework, utility library, verification component library, and simulator independent scripting flow. It includes capability for Constrained Random, Functional Coverage, Scoreboards, FIFOs, Memory Models, error logging and reporting, and message filtering. Capable for Co-simulation and equivalent to SystemVerilog's UVM for VHDL
PyUVM	Universal Verification Methodology implemented in Python instead of SystemVerilog. pyuvm uses cocotb to interact with simulators and schedule simulation events.
UVVM	Universal VHDL Verification Methodology and Library for structured VHDL-based testbenches for reusability, maintainability and extensibility. Features include advance & enhanced randomization, functional & specification coverage, monitors, error injection, scoreboards, transactions, verbosity control, specification coverage, checkers, alert handling and logging.
SVUnit	Test framework for Verilog and SystemVerilog code for ASIC and FPGA developers. It is automated, fast, and lightweight.
VUnit	Unit testing framework for VHDL and SystemVerilog. Usually for continuous and automated testing
Design for Testability (DFT)	
ATALANTA	Automatic test pattern generator for stuck-at faults in combinational circuits
Fault	Automatic test pattern generation for netlists, scan chain stitching, and synthesis scripts. (Difetto (ALPHA))
Fenice	Customizable fault-simulation and gate-level editing library for sequential circuits.
HOPE	Fault simulation and test pattern generation in digital circuits.
Physical Design & Verification (Back-End)	
RTL2GDS / netlist GDS	
OpenROAD FlowScript	Autonomous, open-source tool chain for digital SoC layout generation, focusing on the RTL-to-GDSII phase of system-on-chip design. digital design. Aiming to bring down the barriers of cost, expertise and unpredictability. Addition to open PDK, it also supports proprietary platforms: GF55, GF12, Intel22, Intel16 and TSMC65. It consists of several tools from synthesis to signoff phase of physical implementation.
LibreLane	Extensible digital RTL2GDS flow, successor to OpenLane. It is ASIC infrastructure library based on various components including OpenROAD, Yosys, Magic, Netgen, CVC, KLayout and custom scripts for design exploration and optimization.
Silicon Compiler	Modular build system for hardware. It design languages are C, Verilog, SV, VHDL, Chisel, Migen/Amaranth, Bluespec, MLIR. Various RTL to GDSII tools are supported and can be integrated to run the flow.
mflowgen	Flow management framework for digital ASIC design, automate and orchestrate tool flows from RTL to GDSII, integrating multiple EDA tools in a flexible, modular pipeline.
Synthesis	
Yosys	OpenVerilog RTL synthesis suite, currently supports Verilog-2005, some subsets of VHDL and SV via plugins. Targets the SkyWater 130nm open source PDK for fully open source ASIC design, also perform formal verification. Integrated into many RTL"GDSII flow tools as synthesis tool.
LSOracle	Framework developed on the top of EPFL logic synthesis libraries to unlock efficient logic manipulation by using different logic optimizers. Input:RTL (Verilog/flattened) or BLIF/gate-level netlist from Yosys
Naja eda	Structural Netlist API for EDA post synthesis flow development
Floorplan, Place & Route	
Coriolis	Free database, placement and routing tool for VLSI design. Its main components are the Hurricane database, the Etesian placer and the Katana router.
LibrEDA	Placement algorithms, software framework for the physical design of silicon chips to simplify development of place and route tools.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Qrouter	Multi-level, over-the-cell maze router.
Qflow	Digital synthesis flow using Verilog or VHDL for complete ASIC flow.
RePlAce	Global placement tool with features: analytical and nonlinear placement algorithm, support mixed size placement mode, support fast image drawing modes.
DREAMPlace	Deep learning toolkit-enabled VLSI placement, run on both CPU and GPU. It integrates a GPU-accelerated detailed placer, ABCDPlace. Multi-threaded CPU and optional GPU acceleration support for detailed placement.. LEF/DEF support as input/output. Python binding and access to C++ placements database. Improved efficiency for wirelength and density operators from TCAD extension. Support moveable macro, support time optimization in global placement, integrate OpenTimer for static timing analysis.
CTS	
TritonCTS 2.0	TritonCTS automatically generates a buffered clock tree, used in OpenROAD.
Layout Editor, Physical verification: Netlist (Compare, verification)	
Glade	Glade is GTK+ Layout Design Editor, Gds, Lef And Def Editor – layout and schematic editor, DRC, extraction, LVS.
Klayout	Mask layout tool that GDSII viewer and editor, finla layout verification, support DRC, LVS. Used in many RTL2GDSII tool chains.
Magic	IC layout tool, support extraction, DRC.
IRSIM	Switch-level simulation.
Netgen	Netlist management system: LVS tool for comparing SPICE or Verilog netlists.
CVC	Circuit Validity Checker. Voltage aware ERC checker for CDL netlists.
Power estimation & Timing Analyzer	
OpenSTA	Gate level static timing verifier, analyzer.
OpenTimer	High-Performance Static Timing Analysis Tool for VLSI Systems

2.2.2. Feature comparison

Various available open-source EDA tools that are used in digital system design with their features are listed in the following tables. Digital system design includes many different design and verification stages and requires wide range of EDA tools and framework accordingly.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Table 2.2. System Level Tools Features Comparison.

☑ = Fully supported ⚙ = Limited or indirect support ✕ = Not supported

Features	System Level EDA Tools: System Design & Architectural Exploration							
Simulator, Emulator, Virtual Prototyping								
	FireSim	**Gem5**	**GVSoC**	**QEMU**	**Renode**	**SST**	**SystemC (TLM)**	**VPSim**
ISA Support	⚙ RISC-V. Custom	ARM, x86, RISC-V,POWER SPARC, MIPS	RISC-V	ARM, x86, RISC-V, PPC, SPARC,, MIPS, OpenRISC and more	ARM, RISC-V, x86, POWER, SPARC, Xtensa, MSP430X	⚙	⚙	Arm, RISC-V
Full-System & Sub-System Simulation	☑	☑	☑	☑	☑	☑	☑	☑
Cycle-Accurate	☑	☑	✕	✕	✕	⚙	☑	✕
Extensible / Modular	☑Chisel based	☑	☑	⚙	☑ Co-sim	☑co-sim	☑	☑with QEMU
Use Case	Hardware simulation, development, and IP generation, HW/SW co-design	CPU/memory architecture research	Multicore, multi memory level RISC-V based IoT processor & System	OS bring-up, virtualization, fast regression	Firmware bring-up, embedded HW/SW co-dev	Multi-component heterogeneous system sim	System modeling, virtual prototyping, HW/SW Co-design, RTL Verification	Profiling, benchmarking HPC to multi SoC including heterogeneous multicore architecture
Sub-System Specific	**Muchisim**		**noxim**		**Ramulator**		**SimEng**	**Spike**
ISA Support	⚙		✕		✕		☑ Arm, x86, RISC-V, POWER	RISC-V only
Subsystem Sim	☑ Multi-Chip Multicore		☑NoC		☑ DRAM standards		☑ Pipelines, caches	⚙
Use Case	Multicore CPU, multi-chiplet architecture research		NoC research & interconnect exploration		Memory controller/DRAM architecture research		Pipeline/microarchitecture studies	RISC-V ISA compliance & functional verification
IP Reuse, Integration, Power Modeling & Analysis								

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

	Switchboard	FuseSoC + Edalize	Kactus	McPAT	MESSY
Modeling /Simulation	☑ Connects subsystems: FPGA, RTL, SW models	⚙ (assist)	✗	☑	☑ Simulation (GVSoC), Power modeling (SystemC-AMS)
IP Packaging, reuse, Integration & Doc gen	⚙ Framework to integrate, communicate	⚙	☑	✗	⚙
Power Estimation	✗	✗	✗	☑	☑
Area & Timing Estimation	✗	✗	✗	☑	⚙
IP-XACT Support	✗	⚙ IP reuse, aid creation, build & simulation SoC/ASIC solutions	☑ native IP packaging, reuse and SoC assembly	✗	⚙ Functional sim, system/subsystem level power modeling
Use Case	HW/SW co-sim, connecting simulators /IPs			Power, Area & Timing Analysis	

HLS Framework, Domain–Specific IP Generators (HW/SW Co-design)						
	CIRCT	Dynamatic	PandA-bambu	PipelineC	XLS	OpenASIP
IP /RTL Generator	✗	✗	☑ (HLS)	☑ (HLS)	⚙	☑ Processor generator
Input	MLIR/LLVM IR, FIRRTL, Calyx, Python, Chisel, OpenCL	C/C++ (LLVM)	C/C++	Restricted C	DSLX, C++ IR	Architecture Definition File, C/C++, OpenCL
Output	RTL/IR	RTL (Verilog/VHDL)	RTL (VHDL/Verilog)	Verilog/VHDL	RTL (Verilog/SV)	Custom processor RTL + compiler
Use Case	Design accelerators, support HW design, transformation and generation, formal verification tooling, HLS	Dataflow accelerators	RTL from C/C++, high-level verification, ASIC, parallel HW accelerator design, design flow space exploration	FPGA pipeline, streaming accelerators	Verified HLS, accelerate design	Custom domain-specific ISAs processors

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Table 2.3. Front-End EDA Tool's Features Comparison

Features	Front-End EDA Tools (or Framework): Logic (HDL) Design, Verification & Testing				
HDL Framework, Toolchain, IP Generators					
	Amaranth (nMigen)	Chisel	Chipyard	Py2HWSW	
Host Language	Python	Scala	Chisel	Python	
Generates Synthesizable Verilog	☑	☑	☑	☑	
Parametric/Reusable Design	☑	☑	☑	☑	
Built-in / includes Simulation	☑	✗	☑	☑	
Ecosystem Integration	☑	✗	☑	☑	
Large-Scale ASIC/SoC	❁	☑	☑	☑	
Use Case	Rapid prototyping, FPGA IP blocks, teaching, Not widely used in ASIC flow.	ASIC/SoC research, parameterized CPU/accelerator design. RTL& FPGA accelerated sim, automated VLSI flow	Ecosystem to design & evaluate SoC/ HW	HW/SW codesign, lint-friendly core gen for ASIC/FPGA, flow automation	
Memory Generators					
	Bsg_fakeram	Lake	OpenRAM	RgGen	
Memory Generator	☑	☑	☑	Register generation	
Extensible	❁	❁	☑	☑	
Input	JSON (configuration file)	Macros, high-level specification	Config + PDK	YAML, JSON and more	
Output	RTL behavioral RAM models	RTL/ macro buffers	SRAM GDSII, netlist, Verilog/SPICE models	RTL, SW headers, UVM models	
Use Case	Simulation/verification (Fake SRAM models), Prototyping	Specialized memory/buffer IP	Generate ASIC SRAM macros	Auto-generate register blocks	
Linters					
	PySlint	SVA Lint	svlint	TerosHDL	Verible
Verilog Support	☑	✗	☑	☑	☑
SystemVerilog Support	☑	SVA	☑	☑	☑
VHDL Support	✗	✗	✗	☑	✗
SVA (SystemVerilog Assertions) Linting	☑	☑	❁	☑	❁
Style/ Formatting Checks	☑	❁	☑	☑	☑
Semantic Checks	❁	❁	❁	❁	❁
Custom Rule Support	❁ Via python	☑ Assertion specific	☑	✗	☑
RTL Compiler/Simulator (Optionally Synthesis)					

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

	essent	GHDL	Icarus Verilog	NVC	Surelog	Synlig	Verilator
Verilog Support	✗	✗	☑	✗	⚙	☑	☑
SystemVerilog Support	✗	✗	⚙	✗	☑	☑	Subset
VHDL Support	✗	VHDL-87, -93,2002, partially 2008,2019	✗	☑	✗	✗	✗
Compiled Simulation (fast)	⚙ FIRRTL to C++	☑	✗	✗	✗	✗	☑
Compilation	⚙ FIRRTL to C++	☑	☑	☑	☑	☑	☑
Simulation	☑	☑	☑	☑	✗	✗	☑
Synthesis	✗	⚙		✗	✗	☑	✗
Event-Driven Accuracy	Cycle-accurate	☑	☑	☑	✗	✗	Cycle accurate
Waveform format	FST, VCD	FST, GHW, VCD	FST, LXT, VCD	FST, VCD	✗	✗	FST, VCD
Testing & Verification Framework Integration	✗	OSVVM, UVVM, VUnit, cocotb	Cocotb	OSVVM, UVVM, VUnit, cocotb	✗	✗	Cocotb,C++/SystemC Testbench (UVM in development)
Primary Focus	Chisel/FIRRTL sim	VHDL sim/synth	Verilog compilation & Sim	VHDL compilation & sim	SV Frontend	SV-17 synthesis using Surelog, Yosys	Fast Verilog sim, lint, compile
Waveform Viewer							
	fstdumper	**GtkWave**		**simview**	**Sooty**		**Surfer**
VCD Support	✗	☑		☑	☑		☑
FST Support	☑	☑		☑	✗		☑
LXT, LXT2, GHW, VZT Support	✗	☑			✗		⚙
GUI Waveform Viewer	✗ Via GTKWave	☑		☑	☑		☑ Modern, web interface
Scripting/Automation	⚙	☑		☑	☑		⚙
Primary Focus	Fast dumping of simulation waveform.	General purpose. Debugging Verilog or VHDL simulatio.		Debug & analyze SV HW design.	Lightweight waveform visualization as an alternative viewer.		Lightweight, highly configurable.
Formal Verification							
	ABC	**mcy**	**cvc5**	**AutoSVA**	**AutoCC**	**SymbiYosys (sby)**	**eqy**

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Mixed HDL	☑	❁	❁ SMT-LIB	❁	❁	☑	❁
Formal Property Checking	☑	☑	☑	☑	☑	☑	✗
Automation (Property/Assertion Gen)	✗	✗	✗	❁	☑	❁	✗
Equivalence Checking	☑	❁	☑	✗	✗	❁	☑
Model Checking	☑	✗	☑	❁	✗	☑	✗
Integration with Open-source Tools	Yosys, LSOracle, eqy	Yosys	Yosys, SBY	SBY	SBY	Yosys, Boolector, cvc5	Yosys,SBY
Verification Framework & Testing							
	OSVVM	**cocotb**	**PyUVM**	**OSVVM**	**UVVM**	**SVUnit**	**VUnit**
HDL Support	VHDL	Use Python for VHDL & SV	Python	VHDL	VHDL	Verilog, SV	VHDL, SV
Verification Style		Coroutine based	UVM-like (Python)	Library based	UVM like	Unit Testing	Unit Testing with Framework
Full Verification Methodology	☑	✗	☑	☑	☑	✗	✗
Testbench Automation	☑	❁	☑	☑	☑	✗	☑
Constrained Random or Randomization	☑	☑	☑	☑	☑	✗	❁ via OSVVM
Coverage	☑	❁	☑	☑	☑	❁	☑
Assertion	❁	❁	❁ via SV	VHDL assert	☑	❁	❁
Scoreboard or Checker		☑	☑	☑	☑	✗	❁
Unit Testing	✗	✗	✗	✗	✗	☑	☑
Design For Testing (DFT)							
	Atalanta	**FAULT** (Difetto (ALPHA))	**Fenice**	**Hope**			**DFT in OpenROAD**
Scan Insertion	✗	☑	✗	✗			☑
ATPG	☑	☑	✗	✗ Test pattern generation			✗
Scan Chain Testing	✗	☑	❁ library	❁ fault simulation			✗
MBIST	✗	✗	✗	❁ (BIST)			✗
JTAG Macro	✗	✗	✗	✗			✗
Boundary Scan	✗	✗	✗	✗			✗
Scan-chain compression	✗	✗	✗	❁			✗

Some tools are used in both frontend and backend stages of digital design flows. Therefore, tools such as Yosys for synthesis and static timing analysis tools are described in the physical design section, as they are integrated to the RTL-to-GDSII design flow in some ASIC design flow. Many small

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

or core tools and modules which are integrated inside some RTL-to-GDSII tools are not listed in the tables.

Table 2.4. Comparison of Physical Design and Verification EDA Tool Features

☑ = Fully supported ⚙ = Limited or indirect support ✗ = Not supported

Features	Back-End EDA Tools: Physical Design & Verification					
RTL2GDSII Flow (Autonomous, Custom, Modular)						
RTL2GDS / netlist GDS	OpenROAD FlowScript (ORFS)	LibreLane	Silicon Compiler	mflowgen		
RTL → GDSII Flow	☑	☑	☑	Netlist to GDS		
OpenROAD Based	☑	☑	☑	Can call OpenROAD		
Multi-tool Support	✗	✗	☑	☑		
Customizable Flow	⚙ Fixed	⚙	☑	☑		
Dependency / Reproducibility	Script Based	YAML/JSON Flow	☑	☑		
Cloud/CI Friendly	Manual Script	Can be scripted	☑	☑		
AI/ML	☑	✗	⚙ Possible	✗		
Primary Use Case	Automated Flow	Configurable ASIC Flow	GDSII Framework	Flow construction + reproducibility infra		
Logic Synthesis						
Synthesis	Yosys	LSOracle	Naja EDA			
Verilog Support	☑	☑	☑			
SystemVerilog Support	⚙	☑	⚙			
VHDL Support	Via Plugin	⚙ via Yosys	✗			
Tech Mapping (Liberty)	☑	✗	✗			
Logic Optimization (AIG/MIG, etc.)	☑	☑	✗			
Formal Equivalence Checking	⚙ via SymbiYosys, ABC	⚙ relies on Yosys/ABC	✗			
Extensible / Plugins	☑	☑	☑			
Place and Route (P&R)						
Place & Route	Coriolis	DREAMPLace	LibrEDA	RePlAce	Qflow	Qrouter
Placement	☑	☑	☑	☑	☑	✗
Routing	☑	✗	✗	✗	☑	☑
Clock Tree Synthesis	☑	✗	☑	✗	⚙	✗
Timing Analysis Integration	☑	⚙	✗	⚙	⚙ Call STA externally	✗
Tech Mapping (LEF/DEF/Liberty)	☑	☑	⚙	☑	☑	☑

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Scalability	⚙ Medium size	☑	⚙	☑	✗	⚙ Small, medium
Physical Verification (Layout), Timing Analyzer						
Layout, STA, Netlist	Klayout	Glade	Magic	OpenSTA	OpenTimer	Netgen	CVC
Layout Edit/View	☑	☑	☑	✗	✗	✗	✗
DRC	☑	☑	☑	✗	✗	✗	✗
LVS	☑	☑	☑	✗	✗	☑	✗
Parasitic Extraction	✗	☑	☑	✗	✗	✗	✗
Timing Analysis	✗	✗	✗	☑	☑	✗	✗
Power Estimation	✗	✗	✗	⚙ Switching activity input, limited	basic dynamic power est	✗	✗
Netlist Comparison	☑	✗				☑	☑

Overview of OpenROAD Flow Script (ORFS)

OpenROAD is open-source platform for digital semiconductor design. Its native flow, OpenROAD Flow Scripts (ORFS), is a fully autonomous RTL-to-GDSII implementation framework built on OpenROAD and other open-source tools. It is described here in more detail, since it seems to be developing into a widely adopted flow framework and collection of tools for bringing digital designs from RTL to GDSII. ORFS is designed to enable no-human-in-the-loop (NHIL) digital design, targeting a rapid 24-hour turnaround from RTL to GDSII. It supports rapid architecture and design space exploration, enables early prediction of quality of results (QoR), and provides detailed physical design implementation. Basic Flow stages for a physical design implementation using OpenROAD with respective tools are listed in short in the following steps.

Synthesis

- Generate gate-level netlist (yosys).
- Perform cell mapping (abc).
- Perform pre-layout STA (OpenSTA).

Floorplanning

- Floorplan initialization – Define the chip area, utilization, core area, including of the macros, cell sites and tracks. IO pin and cell placement (ioPlacer, ICeWall).
- Macro placement (RAMs, embedded macros) (Hier-RTLMP, TritonMacroPlacer).
- Tap cell and well tie insertion.
- Generate power distribution network (PDNGEN).

Global Placement

- Standard cell placement.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

- Automatic placement optimization and repair for max slew, max capacitance, and max fanout violations and long wires. (RePlAce)

Detailed Placement

- Perform detailed placement to legalize the globally placed components by aligning to grid, adhere to design rules .
- Incremental timing analysis for early estimates.(OpenSTA)

Clock Tree Synthesis (CTS)

- Insert buffers and resize for high fanout nets.
- Synthesize clock three structure (TritonCTS 2.0).
- Optimize setup/hold timing.

Global Routing

- Antenna repair.
- Perform global routing to generate routing guide files for detailed router (FastRoute).

Detailed Routing

- Perform detail route (TritonRoute).
- Legalize routes, DRC-correct routing to meet timing, power constraints.

Chip Finishing

- Perform SPEF extraction, parasitic extraction using OpenRCX.
- IR drop analysis (PDNSim)
- Final timing verification
- Final physical verification
- Dummy metal fill for manufacturability
- Use Klayout or Magic using generated GDS for DRC signoff.

The OpenROAD build system combines multiple tools into a single executable by compiling each tool as a library and linking them together. It relies on OpenDB for database system and additional open-source tools beyond those listed above.

2.3. Analog/Mixed Signal/RF

This subsection introduces open-source tools used in analog, mixed signal, and RF design flows. Since the design flows share similar steps, the tools are grouped by design step with some modifications where needed.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

2.3.1. Tool descriptions

Table 2.5. Analog/Mixed Signal/Design Tool Descriptions

Tools Description	
Schematic capture EDA tools	
Xschem	A Schematic capture tool well established in the open-source silicon domain
Qucs-S	A schematic capture tool for RF design widely used by RF designers at discrete level
XCircuit	A circuit drawing and schematic capture tool
Qucs	Predecessor of Qucs-S
XIC	A set of tools for IC design, which was used in production and later released to open-source domain.
RevolutionEDA	A new schematic and symbol editor targeting custom integrated circuit design with integrated simulation and plotting capabilities. The software is under Common Clause v 1.0 license which means access to the source code, but some restrictions on further code redistribution
Circuit simulation	
NGSPICE	A SPICE simulator for analog, mixed-signal circuits and OSDI interface support
Xyce	A SPICE compatible simulator from Sandia National Laboratories for large AMS designs
VACASK	A modular simulator supporting Verilog -A models and targeting RF design flow. The tool supports OSDI interface.
Gnucap	A mixed-signal simulator with modular, plugin-based architecture targeting VerilogAMS support
WRSpice	A SPICE compatible simulator from WRCad
Verilog-A model generators	
OpenVAF	A Verilog-A compiler for ngspice and VACASK
ADMS	A Verilog-A model compiler for Xyce and WRspice
Gnucap ModelGen	A VerilogAMS model compiler for Gnucap
Electromagnetic field analysis and simulation	
OpenEMS	An electromagnetic simulator supporting RF and mmWave co-design. Method: FDTD. Limited user interface, model is usually built using Matlab or Python scripts. Some 3rd party plugins to create model scripts from open-source CAD tools. IHP workflow to create and run this model script from GDSII file
AWS Palace	An electromagnetic simulator for RF/mm-wave supporting single computer to large-scale EM analysis and cloud workflows. Method: FEM. Comprehensive support for many features and solver options known from commercial RF FEM tools. Solver engine only, no user interface, model and mesh must be built with other tools. IHP workflow to create these simulator input files from GDSII file.
ElmerFEM	A multi physics software, some support for RF EM modelling. Method: FEM. Solver engine only, no user interface, model and mesh must be built with other tools. Not obvious how to apply to RF models. Limitations for RF solver, e.g. direct solver only (limited model size), no adaptive mesh refinement. IHP workflow is planned (work in progress).
RF-SciKit	A BSD-licensed package for RF/Microwave engineering. It provides a modern, object-oriented library which is both flexible and scalable.
Layout Editing	
Klayout	Layout editor with integrated DRC and LVS capabilities
Magic	TCL/TK based highly scriptable layout editor with DRC capabilities and netlist extraction features
XIC	Layout editor from WRCad tool set
GDSFactory	Python based highly scriptable layout automation tool
RevolutionEDA	Python based integrated design environment

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

DRC engines	
Klayout	Built in features to implement DRC rules. It includes DRC markup browser
Magic	It includes a DRC engine. It can be run in a live mode (while editing the layout)
XIC	DRC engine included in XIC tool set
GDSFactory	Python based DRC engine
LVS engines	
Klayout	Built in features to implement LVS rules. It includes LVS markup browser
netgen	A netlist comparison tool. It supports SPICE and CDL formats.
XIC	LVS engine included in XIC tool set
GDSFactory	Python based LVS engine
Integrated solutions	
ORDeC	(Open Rapid Design Composer) is an open-source **custom IC design platform**. Its goal is to provide an accessible and streamlined interface to design and analyze analog, mixed-signal and custom digital integrated circuits from schematic to layout.
Parasitic Extraction	
Klayout - PEX	Klayout based wrapper for parasitic extraction from a layout
Magic	Layout tool with the capability of generating a netlist with parasitic from a custom layout.
OpenEMS	EM field solver capable to determine S-parameters of an element, which then can be identified as parasitic element
FastCap	A multipole-accelerated capacitance extraction program
FastCap2	3D capacitance solver
FastHenry	A multipole-accelerated inductance analysis program.
Circuit Documentation	
CACE	A tool for automatic generation of documentation based on automated testbenches.
Waveform Viewers	
GTKWave	Analog and digital signals waveform viewer
GAW	GTK analog wave viewer
Surfer	An Extensible and Snappy Waveform Viewer
ngspice	A native plotting backend for ngspice
Xschem	A Xschem built-in plotting objects
Qucs-S	Qucs-S built in plotting objects
gnuplot	A generic and versatile plotting tool
XIC	XIC plotting backend from WRCad tool set

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

2.3.2. Feature comparison

Table 2.6. Analog/Mixed Signal/Tool Feature Comparisons: Schematic editing

☑ = Fully supported ⚙ = Limited or indirect support ✕ = Not supported

Features	Analog design EDA Tools: Schematic editing				
	Xschem	Qucs-S	XIC	Qucs	RevEDA
Custom symbol definition	☑	☑	☑	☑	☑
Spice netlisting	☑	☑	☑	☑	☑
Verilog netlisting	☑	⚙	✕	⚙	✕
Spectre netlisting	☑	⚙	✕	✕	✕
Callbacks for instance parameters calculation	☑	☑	✕	✕	✕
Multiple simulator support	☑	☑	✕	☑	✕
Cell view selection and configuration (schematic, extracted, EM, model, etc.)	✕	☑	☑	☑	☑
Hierarchical schematics	☑	☑	☑	☑	☑
Version control support	✕	✕	✕	✕	✕
Multiuser support (editable, read-only)	✕	✕	✕	✕	☑

Table 2.7. Analog/Mixed Signal/Tool Feature Comparisons: Simulators

☑ = Fully supported ⚙ = Limited or indirect support ✕ = Not supported

Features	Analog design EDA Tools: Simulators				
	NGSpice	Xyce	VACASK	Gnucap	WRSpice
OP – operational point	☑	☑	☑	☑	☑
DC – dc dweep analysis	☑	☑	☑	☑	☑
TRAN – transient analysis	☑	☑	☑	☑	☑
AC – small signal analysis	☑	☑	☑	☑	☑
Parameter Sweep	indirect	☑	☑	indirect	☑
SP – scattering parameter analysis	☑	Via Linear analysis	✕	✕	✕
NOISE – small noise analysis	☑	☑	☑	☑	☑
SENS – sensitivity analysis	☑	☑	✕	✕	☑

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Harmonic Balance – nonlinear AC analysis	❁	☑	☑	✗	✗
TR transfer function	☑	☑	☑	✗	☑
AMS simulation	Via XSPICE	Via custom code	✗	Native VerilogAMS	Native Verilog
PZ – Pole Zero analysis	☑	✗	✗	✗	☑
DISTO – Distortion analysis	☑	✗	✗	✗	☑
PSS – Periodic Steady State	❁	✗	✗	✗	✗
PAC – Periodic AC analysis	✗	✗	✗	✗	✗
Touchstone format support	Write	Write	✗	✗	✗
OSDI support	☑	✗	☑	✗	✗
Measurements support	☑	☑	✗	☑	☑
Fourier analysis	☑	☑	☑	☑	☑
Temperature analysis	☑	☑	☑	☑	☑
Output formats	RAW and text	RAW and text	RAW and text	text	RAW and text
Phase noise	✗	✗	✗	✗	✗
Transient noise	Scripted	✗	✗	✗	✗
Statistical analysis	Scripted	Native command	Scripted	Scripted	Native command
Corner / multi-corner simulation	Single corner at once	Single corner at once	Single corner at once	Single corner at once	Single corner at once
Model support (Verilog, Verilog-AMS, etc.)	Via OSDI	Via ADMS	Via OSDI	Via Gnucap-Model Generator	Via ADMS

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Table 2.8. Analog/Mixed Signal/Tool Feature Comparisons: Layout Editors

☑ = Fully supported ⚙ = Limited or indirect support ✕ = Not supported

Features	Analog design EDA Tools: Layout Editors				
	Klayout	Magic	XIC	GDSFactory	RevEDA
Parametric cell support with PyCellStudio	☑	✕	☑	✕	✕
Parametric cell support with TCL	✕	☑	✕	✕	✕
Parametric cell support with Python	☑	✕	✕	☑	☑
DRC engine	☑	☑	☑	☑	Wraps around klayout
LVS engine	☑	Netlist extraction	☑	☑	Wraps around klayout
PEX engine	☑	☑	☑	✕	✕
Hierarchical layout support	☑	☑	☑	☑	☑
Multiuser support (editable, read-only)	✕	✕	✕	✕	☑
Back annotation between schematic and layout	✕	✕	✕	✕	✕

Table 2.9. Analog/Mixed Signal/Tool Feature Comparisons: Mixed Signal

Features	Analog design EDA Tools: Mixed signal design
	Support level
Supported AMS hardware description languages (HDLs)	Gnucap supports Verilog-AMS, ngspice supports XSpice code models
AMS HDL integrated development environment (IDE) editor	✕
AMS HDL compilation as a shared library	Supported by gnucap-model-generator Verilog-AMS LRM 2.4.0
Automatic Analog-Digital netlist partitioning	No, however gnucap supports manual partitiong
Event-based and electrical co-simulation	Gnucap supports this feature, also ngspice can run analog on top simulation, which contains digital blocks (compiled from Verilog)
Analog-Digital waveform viewer	GTKWave, Surfer, lack of interfaces for analog waveforms
AMS IP modeling for digital-on-top integration	LEF views can be generated using Magic VLSI, There are several tools: lc-time, libretto, CharLib, which can characterize timing for analog cells and generate Liberty files.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Table 2.10. Analog/Mixed Signal/Tool Feature Comparisons: EM Simulators

Features	Analog design EDA Tools: EM Simulators		
	OpenEMS	**AWS Palace**	**ElmerFEM**
EM Solver			
Solver Type (FDTD, FEM, MoM)	FDTD	FEM	FEM
Frequency Range Supported	DC to THz	No upper limit, low frequency limit for RF solver depends on mesh size, typ. 10 MHz, DC by extrapolation. Static solvers available but not included in IHP workflow.	No upper limit, low frequency limit depends on mesh size, to be determined, DC by extrapolation
Selectable accuracy solver precision (fast, high accuracy)	Not really (only energy convergence limit)	Yes, choose order of FEM basis function	Yes, choose order of FEM basis function
Use case orientation (IC-level, package level, antenna)	RF at IC level and package level and antennas. Often slower than Palace FEM, but requires less RAM for large models than FEM -> possibly better for some chip-on-package work. Mesh less efficient for diagonal polygons, best for Manhattan shapes.	RF at IC level and package level, antenna far field is new feature and needs to be tested. Using FEM, mesh is more efficient than FDTD for models with dimensions ranging from large to very small. However, FEM often requires more RAM than FDTD, antenna models might run into memory limit. Large models possible using MPI on data center scale. Future speed-up possible if mesh algorithm in IHP workflow (gmsh) can be improved.	Multiphysics otherwise see Palace. Complexity is limited because at present, only direct matrix solver is available for the RF solver. Work on RF solver is rather new, work in progress. Future speed-up possible if mesh algorithm in IHP workflow (gmsh) can be improved.
Multi-threading	Yes, but does not scale well for more than 4 threads	Yes	Yes
Load sharing / Data center	No	MPI	MPI
Recommended RAM size	16 GB	64 -128 GB	64 -128 GB
Geometry and Materials			
Support for planar and 3D structures	Yes	Yes	Yes
Fully customizable layer stack (metal thickness and conductivity, vias, substrate)	Yes	Yes	Yes, but at present metal resistance goes to zero towards DC. TO DO: implement wideband metal loss model.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Meshing			
Automatic meshing	Yes, by IHP workflow script (custom code)	Yes, by IHP workflow script (gmsh + custom code)	Yes, by IHP workflow script (gmsh + custom code)
Adaptive meshing (shape and current density based)	Not used in FDTD	Yes, adaptive mesh refinement option during solve	No adaptive mesh refinement during solve
Simulation Features			
DC and low frequencies	Solution goes down to DC, but results towards DC might have some ripple, depending on energy convergence limit.	Static solver available but not implemented in IHP workflow script. IHP workflow script uses extrapolation to DC from low frequency data (10 + 20 MHz). This DC data is more accurate and robust than openEMS FDTD data.	IHP workflow script uses extrapolation to DC from low frequency data (10 + 20 MHz)
Adaptive frequency sampling	FDTD always gives smooth wideband sweep from FFT	Yes, robust and fast	Very limited
Multi/differential-ports and multi-signal analysis	Yes	Yes	Yes
S-parameter extraction	Yes	Yes	Yes
Field visualization (E/H fields)	Optional (Paraview)	Optional (Paraview)	Optional (Paraview)
Current density visualization	Optional (Paraview)	Optional (Paraview), somewhat difficult to use and see	?
Usability			
GUI available	openEMS is essentially a solver with API, but provides a model viewer application with limited editing capabilities. GUI for IHP workflow under development for simple/typical use cases (standard port, no multi-technology config) 3rd party interface for FreeCAD solid modeler available (no GDSII, not automatic meshing)	Palace is a pure number cruncher. IHP workflow shows model & mesh in gmsh GUI. GUI for IHP workflow under development for simple/typical use cases (standard port, no multi-technology config)	Elmer is essentially a number cruncher. ElmerGUI tool is not compatible with text-based Elmer configuration used by IHP workflow. IHP workflow shows model & mesh in gmsh GUI. GUI for IHP workflow under development for simple/typical use cases (standard port, no multi-technology config)
Scripting for automation	IHP workflow uses openEMS Python API, openEMS also supports Matlab/Octave API.	IHP workflow uses Python with gmsh to create model, then calls FEM solver	IHP workflow uses Python with gmsh to create model, then calls FEM solver
Waveform viewer	3rd party by script	3rd party by script	3rd party by script

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Input format	openEMS API to create models in Python or Matlab/Octave syntax With IHP workflow: GDSII layout and XML stackup	Palace requires FEM mesh (typically from gmsh) and solver config file. With IHP workflow: GDSII layout and XML stackup	Elmer requires FEM mesh (typically from gmsh) and solver config file. With IHP workflow: GDSII layout and XML stackup
Output format	Time domain voltage & current, S-parameters, optional Paraview visualization files	S-parameters, optional Paraview visualization files	S-parameters, optional Paraview visualization files

2.4. Integrated photonics

This subsection introduces open-source tools used in PIC design flow. Many of these tools were originally developed for Analog or RF design, and are used in photonics as if (at different frequency), or with additional dedicated models. Some of the steps may use general-purpose tools like python and Octave which we do not mention here. However, dedicated packages and libraries are included. In some of the steps there are no open-source tools available, which is discussed later in the gap analysis section.

Besides functionality features, it is important to understand and consider factors related to the open-source nature of the software. These factors are outlined in the last part of this section.

2.4.1. Tool descriptions

The "Device-level simulation" category groups multiple tools for electro-magnetic simulations: mode solvers, FEM-, FD-, EME, FMM, etc. based tools, etc. Some of them also allow multi-physics simulations, which is useful for packaging or thermal simulations. Several tools are general EM-simulation tools, while others are dedicated to optical domain.

Table 2.11. Integrated Photonics Tool Descriptions

Tools Description	
Device-level simulation	
FEM-based simulators	
FEMWELL	FEMWELL is a physics simulation tool that utilises the Finite Element Method (FEM).
Elmer FEM	Elmer is a tool that can solve a large number of partial differential equations making it an ideal tool for multiphysical problems.
Palace	Parallel finite element code for full-wave 3D electromagnetic simulations in the frequency or time domain, using the MFEM finite element discretization library and libCEED library for efficient exascale discretizations.
Netgen/NGSolve	Multi-purpose finite element library.
JAX-FEM	JAX-FEM is Automatic Differentiation (AD) + Finite Element Method (FEM),

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Other simulators	
MPB	MPB is a free and open-source software package for computing the band structures, or dispersion relations, and electromagnetic modes of periodic dielectric structures, on both serial and parallel computers.
EMpy	ElectroMagnetic Python is a suite of algorithms widely known and used in electromagnetic problems and optics: the transfer matrix algorithm, the rigorous coupled wave analysis algorithm and more.
FDTDX	FDTDX is an efficient open-source Python package for the simulation and design of three-dimensional photonic nanostructures using the Finite-Difference Time-Domain (FDTD) method.
meep	Meep is a free and open-source software package for electromagnetics simulation via the finite-difference time-domain (FDTD) method spanning a broad range of applications.
meow	A simple electromagnetic EigenMode Expansion (EME) tool for Python.
Tiny3D	Tidy3D is a software package for solving extremely large electrodynamics problems using the finite-difference time-domain (FDTD) method. It requires subscription to the server, but some of the components (e.g. Mode-solver) are open.
EMEPy	An open-source tool for simulating EM fields in Python using the Eigenmode Expansion Method (EME).
FMMAX	FMMAX is an implementation of the Fourier Modal Method (FMM).
Circuit-level design	
Circuit simulation	
SAX	S-Matrices with Autograd and XLA - a scatter parameter circuit simulator and optimizer for the frequency domain based on JAX.
Simphony	A simulator for photonic circuits, is a fundamental package for designing and simulating photonic integrated circuits with Python.
scikit-rf	A python package for RF/Microwave engineering
Lcapy	A Python package for linear circuit analysis. It uses SymPy for symbolic mathematics.
PySpice	PySpice is a Python module which interface Python to the Ngspice and Xyce circuit simulators
VACASK	VACASK (Verilog-A Circuit Analysis Kernel) is an analog circuit simulator. VACASK uses the OpenVAF-reloaded Verilog-A compiler for building the device models as shared libraries.
Qucs-S	Qucs-S is a circuit simulation program based on Qucs circuit simulator. The purpose of the Qucs-S project is to use free circuit simulation kernels (Ngspice, Qucsator, Xyce) with the unified GUI based on Qt6 toolkit.
Phisim	Simulation software package for semiconductor integrated optical circuits. The simulation is a travelling wave finite time difference simulation of an InP based optical integrated circuit.
Layout tools	
Nazca Design	Python based highly scriptable layout automation tool
GDSFactory	Python based highly scriptable layout automation tool
KLayout	Layout editor with integrated DRC and LVS capabilities
gdstk	Gdstk (GDSII Tool Kit) is a C++ library for creation and manipulation of GDSII and OASIS files.
Physical layout verification	
DRC, LVS tools	
KLayout	Built in features to implement DRC / LVS rules. It includes DRC and LVS markup browser
GDSFactory	Python based DRC and LVS engine

2.4.2. Feature comparison

The tables in this section provide functionality comparisons for tools in different categories.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Table 2.12. Integrated Photonics Tool Feature Comparison: Device-level simulation ½

☑ = Fully supported ⚙ = Limited or indirect support ✕ = Not supported

Features	Device-level simulation				
	FEM-based See also comparison table in section 2.2.				
	FEMWELL	Elmer FEM	Palace	Netgen/NGSolve	JAX-FEM
Main purpose	Photonics-focused FEM toolkit	General multiphysics FEM solver	Large-scale electromagnetic simulation	Flexible research FEM framework	Differentiable FEM & optimization
Typical use cases	Waveguide modes, photonic components	Multiphysics engineering	RF/photonic cavities, resonators	Research FEM, custom PDEs	ML-coupled FEM, optimization
Primary domain	EM (wave optics), thermal, electrostatics	EM, thermal, mechanics, fluids	EM (Maxwell)	General PDEs (incl. EM)	Mechanics, thermal, inverse problems
Full-wave 3D EM	⚙	☑	☑	⚙	✕
Time-domain support	⚙	⚙	☑	☑	✕
Frequency-domain support	☑	☑	☑	☑	☑
Eigenmode solvers	☑	☑	☑	☑	☑
Waveguide boundary conditions	☑	⚙	☑	☑	☑
Multiphysics coupling	⚙	☑	⚙	☑	⚙
Integrations	GDSFactory	GDSFactory	GDSFactory	✕	✕
License	GPL-3.0	GPL-2.0	Apache-2.0	LGPL-2.1	GPL-3.0

Table 2.13. Integrated Photonics Tool Feature Comparison: Device-level simulation 2/2

Features	Device -level simulation							
	Other See also comparison table in section 2.2.							
	MPB	EMpy	FDTDX	Meep	meow	Tiny3D	EMEPy	FMMAX
Simulation method	Planewave eigenmode solver	Transfer matrix method, RCWA	FDTD	Multiple, incl. DTFT, mode decompositions, etc.	EME		EME	RCWA
Domain	Frequency	Frequency	Time	Time	Frequency	Frequency	Frequency	Frequency
Output data format	HDF5			HDF5				
Documentation	Good	Limited	Good	Good	Good		Limited	Limited

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Acceleration	Multi-core		JAX-based	Multi-core				JAX-based
Integrations	GDSFactory			GDSFactory	GDSFactory, SAX, Tiny3D	GDSFactory		
License	GPL-2.0	MIT	MIT	GPL-2.0	Apache-2.0	See "Other"	MIT	MIT
Other						Part of a paid software suite		

Table 2.14. Integrated Photonics Tool Feature Comparison: Circuit simulation

☑ = Fully supported ⚙ = Limited or indirect support ✗ = Not supported

Features	Circuit simulation See also comparison table in section 2.2.							
	SAX	**Simphony**	**scikit-rf**	**Lcapy**	**PySpice**	**VACASK**	**Qucs-S**	**Phisim**
Simulation methods	S-parameters	S-parameters	S-parameters		SPICE	SPICE	SPICE	TW-FTD
Application domain	Photonic / RF	Photonic / RF	RF		Electronic circuits	Analog circuits	RF / analog circuits	Photonics, incl. amplifiers
Symbolic	✗	✗	✗	☑	✗	✗	✗	✗
Data formats	Netlist / API	Python API	Touchstone, Z/Y/ABCD, N-port definitions	Netlist	Spice netlist	Verilog-A		Custom, python API
Integrations	GDSFactory	SiEPIC and SiPANN						
License	Apache-2.0	MIT	BSD-3-Clause	LGPL-2.1	GPL-3.0	GPL-3.0	GPL-2.0	GPL-3.0
Other			General-purpose library				Simulation with backends	

Table 2.15. Integrated Photonics Tool Feature Comparison: Layout tools

☑ = Fully supported ⚙ = Limited or indirect support ✗ = Not supported

Features	Layout tools			
	Nazca Design	**GDSFactory**	**KLayout**	**gdstk**
Parametric cell support with PyCellStudio	✗	✗	☑	✗
Parametric cell support with Python	☑	☑	☑	☑
DRC engine	☑	☑	☑	✗
LVS engine	✗	☑	☑	✗
PEX engine	✗	✗	☑	✗
Hierarchical layout support	☑	☑	☑	☑

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Multiuser support (editable, read-only)	✗	✗	✗	✗
Back annotation between schematic and layout	✗	✗ (paid version)	✗	✗
License	GPL-3.0	MIT	GPL-2.0	BSL-1.0

Table 2.16. Integrated Photonics Tool Feature Comparison: Physical Verification

☑ = Fully supported ⚙ = Limited or indirect support ✗ = Not supported

Features	Physical verification	
	KLayout	GDSFactory
DRC	☑	☑
LVS	☑	☑
Scripting language	Custom	Python
GUI mode	☑	✗ (paid version)
Batch mode	☑	☑
Markup browser	☑	✗ (paid version)

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

3. Process design kits

The availability of tools, whether proprietary or open source, does not by itself constitute a design environment. Since the objective of the design process is the implementation of a functional hardware block, it is inherently tied to a specific technological platform selected for this purpose. The link between design tools and the fabrication process is provided by the **Process Design Kit (PDK)**, which consists of a set of libraries compatible with the design tools and exposes the underlying technology to the designer. A PDK is typically delivered together with process specifications, layout rule manuals, library-specific documentation, setup guidelines, and design examples.

Traditionally, PDKs are not publicly available and are distributed to customers under non-disclosure agreements (NDAs). In 2020, for the first time, an open-source PDK—the **Google–SkyWater PDK**—was released under the permissive **Apache 2.0** license. This PDK targets the **SKY130A/B** process of the SkyWater foundry and represents the first manufacturable PDK made available to the open-source community. Following the release of SKY130, **GlobalFoundries** opened access to process information by publishing the **GF180MCU A/B/C** PDKs one year later. In 2023, **IHP** followed with the publication of the **IHP Open PDK** for the **SG13G2** process. More recently, in late October 2025, the Chinese foundry **ICSprout** released partial information about its **55 nm CMOS** process. This section provides a brief overview and comparison of the manufacturable and non-manufacturable PDKs currently available within the open-source ecosystem.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Table 3.1. Open source PDKs.

☑ = Fully supported ⚙ = Limited or indirect support ✗ = Not supported

Features	Manufacturable open source PDK comparison			
	SKY130	GF180	IHP-SG13GS	ICSprout55
CMOS compatible	☑	☑	☑	☑
CMOS minimal feature	150 nm	280 nm	130 nm	55 nm
Standard cells	☑	☑	☑	☑
IO cells	☑	☑	☑	☑
SRAM	☑	☑	☑	✗
Analog primitives	☑	☑	☑	✗
Triple well	☑	☑	☑	✗
Maximum gate voltage	20	10	3.5	✗
BJT devices	☑	☑	Lateral	✗
HBT devices	✗	✗	☑	✗
LDMOS devices	✗	☑	⚙	✗
Diodes	☑	☑	☑	✗
Varactors	☑	✗	☑	✗
Photodiode	☑	✗	✗	✗
Diffusion resistors	☑	☑	✗	✗
Polysilicon resistors	☑	☑	☑	✗
TFR resistors	✗	✗	⚙	✗
MiM Capacitor	☑	☑	☑	✗
Resistive RAM	☑	✗	⚙	✗
BEOL	1 local interconnect 5 metal layers	5 thin 1 thick metal layers	5 thin 2 thick metal layers	5 thin 2 thick metal layers
Available via MPW services	☑	☑	☑	⚙

Apart from the manufacturable PDK's there are some non-manufacturable (predictive) ones mainly used in education. At mature nodes, the OSU standard cell library and the NanGate45 library, both based on the FreePDK45 technology, provide realistic CMOS standard cells widely used for education, benchmarking, and tool development. The FreePDK45 has also a 3D IC version of the

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

original FreePDK45. For advanced-node research there are two alternatives: (1) FreePDK15 PDK and (2) ASAP7 PDK, which offers a predictive, foundry-agnostic 7 nm FinFET technology with multiple standard-cell architectures, enabling exploration of modern scaling challenges such as restrictive design rules and track-based layouts. Together, these open PDKs form a representative stack for studying digital design methodologies across technology generations.

The manufacturable and open source PDKs, namely **SKY130, GF180, SG13G2, ICSprout55** are generally compatible with the open-source tools discussed in Section 2. Owing to the highly structured and automated nature of digital design, well-established standards exist for abstract PDK data. Accordingly, foundries typically provide digital libraries (often referred to as *views*) in standardized formats such as **CDL, LIB, LEF, SPICE, GDS, and Verilog**.

In contrast, within the **analog, AMS, and RF** domains, standardization is considerably less mature, and PDK data formats are often specific to individual design tools. Schematic diagrams and symbols are tool specific and not portable between the tools. Although SPICE language is well established, the way that the open-source simulators handle it is different, which produces inconsistencies between the libraries for different tools. Recent advances in **Verilog-A** modeling support within the open-source ecosystem have contributed to reducing inconsistencies among behavioral models. A similar trend applies to **Python-based parametric cells (PyCells)**, which provide a portable mechanism for describing layout generators across both open-source and proprietary tools, such as **PyCellStudio, KLayout, and XIC**.

Despite these improvements, a **significant portion of technology information is still represented in non-standard, tool-specific formats**. This lack of standardization is particularly evident in **physical verification data**, including **design rule checking (DRC)** and **layout versus schematic (LVS)** rule decks, where rules must be implemented and maintained using proprietary or tool-dependent rule languages. A comparable situation exists for **parasitic extraction** and **electromagnetic (EM) simulation**, in which the absence of a common representation results in multiple, tool-specific descriptions of identical technological data. The last observation produces a significant overhead of work related to the PDK information maintainability.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

4. Gap analysis

The section provides a gap analysis between the proprietary EDA software and open-source equivalents. It is divided into three major subsections, which embrace design domains and technologies. In general, the open-source landscape can be characterized as very dynamic and fragmented in the opposition of conservative and consolidated solutions offered by the vendors. Since 2020, when SKY130 PDK was released, the open-source tools have progressed substantially and follow a constant growth trend. Many existing gaps have been filled, and many are in the development phase. Thus, this document is a snapshot of the landscape now (Q4, 2025). The detailed comparison applies also to the technology nodes, which were released to the open-source domain: SKY130, GF180, IHP-SG13G2, ICSprout55. Since only using these nodes, it is feasible to perform a full design process.

4.1. Digital

Designing digital ICs with open-source tools has become a viable option in recent years, as there are relatively mature system level modeling simulators and emulators that are used in both academia and industry. Recent years have seen growing interest in logic synthesis and physical design, and open-source tools have evolved rapidly and have resulted in multiple chip tape-outs. The following provides a more detailed analysis of the digital design flow stages.

System Level Design

For system level modeling, there exist well-established and widely adopted open-source tools. Emulators (QEMU) allow running operating systems on a variety of target architectures, and functional simulation allows modeling of platforms and peripherals (Renode) as well as microarchitectural details of components (gem5) in system level simulations.

Recent years have seen increasing adoption of open-source RISC-V processor cores into commercial designs. They have been implemented as host cores capable of running linux-based operating systems, as well as accelerators by adding customized hardware for specific tasks. Currently there exist multiple open-source toolsets that can be used to design, customize, generate HDL, and compile programs for RISC-V based cores.

HLS Tools

Existing open-source high-level synthesis tools (Bambu, CIRCT, Dynamatic) that convert high-level program descriptions to RTL implement most of the necessary features to generate hardware circuits of sufficient complexity. The largest gap between the commercial and the current open-source HLS tooling relates to the extent of the commercial IP libraries, verification support and integration of HLS-generated designs to the rest of the development ecosystem. To help the open-

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

source HLS projects to catch up to the commercial tooling, the effort should be focused on the maintainable long-term development of open-source tooling. This enables the accumulation of development effort in the open-source ecosystem, instead of being split among independent projects.

Current actively maintained open-source HLS tools work with either (pragma-assisted) C/C++ input languages or custom HLS-domain-specific languages (**SystemC, DSLX**). In terms of input languages, the current HLS tools can be improved to support parallel programming models to conveniently describe parallel accelerators. Support for portable heterogeneous programming models based on open standards such as **OpenCL** or **SYCL** within an open-source HLS tool could enable end-to-end acceleration flows for FPGAs while also serving as a portable input to an ASIC-synthesis flow. This would also enable system-level simulations and verification to be performed early using portable OpenCL applications.

Logic Design and Synthesis (Frontend)

Open-source synthesis tools have become a viable option in recent years for implementing digital ICs. There are multiple frameworks using **Yosys**, which has become the open-source de facto synthesis tool. Although capable of producing valid synthesis results, it has limitations regarding optimizations and supporting testability. A gap in the tool is the lack of timing-driven synthesis, as the tool does not aim to optimize design timing.

To enable large-scale chip production testing, DFT tools should be developed further. While **Yosys** lacks capabilities for DFT insertion, there are experimental tools being developed that aim to address this gap. **Difetto** enables scan chain and JTAG insertion as well as automatic test pattern generation but is currently confirmed to only work with SKY130. It is also missing advanced DFT features such as BIST IP insertion. A less significant gap in **Yosys** is the lack of support for VHDL or SystemVerilog packages.

By having frameworks or design flow managers such as **LibreLane** or **ORFS**, the open-source EDA tooling seems to have an advantage over commercial tooling, as the automated process can allow a lower barrier of entry into ASIC design. This can also potentially cut development time and costs.

Physical Design (Backend)

Recent years have also seen development of open-source tools for producing GDSII from RTL descriptions. Namely **OpenROAD** and **LibreLane** have been gaining attention as complete tool flows with automation capabilities for the full flow. Both frameworks have been used to successfully tape out designs. While we didn't identify apparent gaps in terms of features in the tool flows, our hands-on experiments showed that they result in significantly larger area footprint on some designs when compared to commercial tooling. For the popular CVA6 core, the results produced by open-source

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

and commercial tools were close to each other, but for a statically scheduled wide-issue DSP core the area footprint was significantly larger.

A limitation for adopting open-source tooling for implementing advanced devices can be the range of support for advanced commercial process nodes. **OpenROAD** lists support for the following proprietary nodes: GF55, 12nm, Intel22, Intel16 and TSMC65.

Result of RTL-to-GDSII Flow with ORFS (OpenROAD Flow Scripts)

To evaluate quality of results (QoR), the RTL-to-GDSII implementation of the CVA6 core from the **ORFS** example flow, along with a custom statically scheduled wide-issue DSP core generated using **OpenASIP** was performed using **ORFS**, an autonomous EDA flow built on the **OpenROAD** toolchain. Both designs were implemented using the **ASAP7** and **SkyWater 130 nm (sky130hd)** process design kits (PDKs) under identical design constraints with flat and hierarchical synthesis. The synthesis results from both open-source **ORFS** tools and commercial tools are shown in the following table.

Table 4.1. Synthesis Results.

Design	CVA6		Custom (TTA) Core	
Tool	ORFS	Commercial	ORFS	Commercial
Utilization (%)	72	70	76	70
Area (μm²)	21746	-8.3 %	2697	-31.7 %
Gate Equivalent (GE)	~ 249 KGE		~ 31 KGE	
GE refers to a unit of measure to specify manufacturing technology-independent complexity of digital circuits. It represents an area of a two-input NAND gate with minimum strength.				

Table 4.1 presents a comparison of synthesis results for area and gate counts obtained using ORFS and commercial tools. **ORFS** uses **Yosys** for synthesis. The synthesis results show that the area and gate count are higher in **ORFS** synthesis compared to commercial tool. For CVA6 core, the differences are relatively small. However, for custom TTA core, the commercial tool results in over 30 % smaller area.

Table 4.2 lists the runtime and memory consumption at various stages of **ORFS** flow with both hierarchical and flat implantation in **ASAP7** and **sky130hd** technology nodes.

Table 4.2. ORFS Flow Results.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Design	Synthesis Imp.	Runtime/ Memory	Synthesis	Floorplan	Place	CTS	Route	Final /Signoff	Total Runtime / Peak Memory
CVA6	Hierarchical (ASAP7)	Runtime (s)	713	224	1040	947	6911	386	18744
		Memory (MB)	914	1134	2080	3919	54780	4308	54780
	Flat (ASAP7)	Runtime (s)	389	217	714	243	2092	252	3909
		Memory (MB)	2220	1010	1679	1402	44597	3074	44597
Custom (TTA) Core	Hierarchical (ASAP7)	Runtime (s)	33	29	234	38	236	67	637
		Memory	192	293	880	394	8844	759	8844
	Flat (Asap7)	Runtime (s)	14	14	320	19	708	110	1185
		Memory (MB)	204	310	767	427	7901	779	7901
	Hierarchical (Sky130hd)	Runtime (s)	53	14	286	49	2234	49	268
		Memory (MB)	110	230	435	279	5274	661	5274
Unit of time (s) = Second									

Figure 4 shown below shows the runtime and memory consumption at various stages of the OpenROAD flow. Based on the obtained ORFS flow results from running the flow for both designs across both technology nodes, with various configurations and synthesis implementation, it can be observed that the routing stages have the highest runtime and memory consumption. Additionally, hierarchical implementations exhibit higher runtime and memory usage compared to non-hierarchical implementations for the same design. The figure also indicates slightly increased runtime and memory consumption during the clock tree synthesis (CTS) in hierarchical implementations of the CVA6 design. However, noticeable such results were not observed during CTS in the custom TTA core as its design size is comparatively small.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

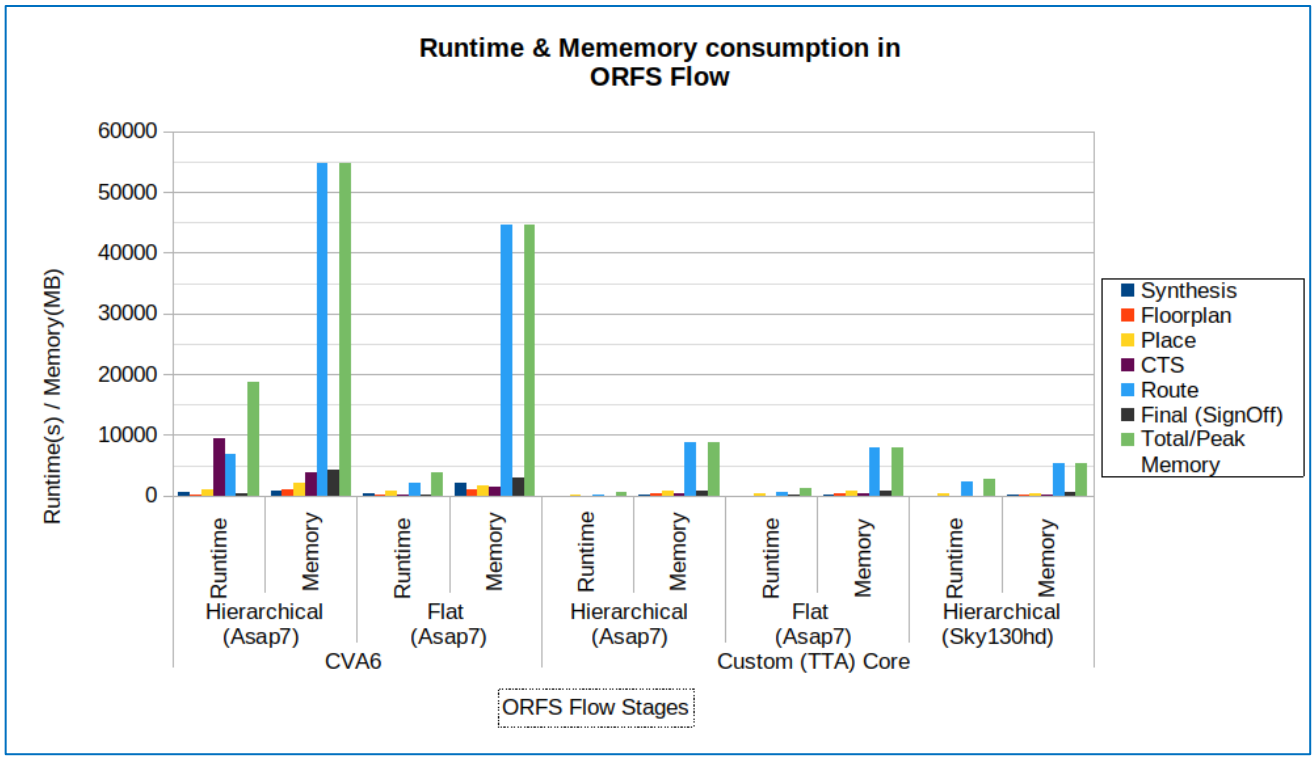

Figure 4. Runtime and memory consumption in ORFS Flow at various stages.

The primary goal was to identify gaps in tool performance, functionality, and efficiency compared to commercial EDA tools, supporting a structured gap analysis. Physical design flow experiments evaluated QoR metrics, including timing, area, and power, alongside runtime and memory consumption, providing insights into the computational efficiency, resource utilization, and scalability of ORFS for both CVA6 and custom DSP designs across different technology nodes. The flow was executed using both earlier and more recent versions of the OpenROAD flow scripts. It was observed that the later versions improved design area and tool efficiency compared to previous runs. However, despite these improvements and promising future, there remain some limitations and areas for further improvements in the ORFS flow, including:

- Supports Verilog only, limited support for SystemVerilog and VHDL designs via plugins.
- Synthesis results, such as cell count and area, are worse compared to commercial EDA tools.
- Unable to target a specific clock frequency by adjusting the area accordingly.
- Provides limited control over the ORFS flow. In addition to that, for example, if fixes are required during detailed routing, the flow cannot be resumed from that stage and must be rerun from the beginning.
- Incremental adjustments between the flow stages, especially during later stages, are not supported, since most of the issues encountered were during routing stage due to congestion issues.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

- Runtime and memory usage increase significantly with even slight increases in design complexity, particularly during the routing stage.
- Compared to commercial EDA tools, the GUI functionality is limited. The GUI does not support interactive component placement or other edits required for error fixing. All such actions must be performed through the Tcl command window of GUI.
- Congestion issues were observed during detailed routing for some designs, although some improvements were observed in later versions of **ORFS**.
- Possible limitations in terms of reliability of formal verification, DRC, ERC checks, and other physical verification which might require more testing.

Hence, it remains necessary to run designs of varying scales and types, applying different constraints and parameters, to conduct a detailed analysis of the **ORF**S flow. Preliminary results indicate that constant updates show some improvements in the tool. However, there still exist various limitations compared to commercial EDA tools.

Result of DFT in OpenROAD

OpenROAD currently supports only scan chain insertion as part of DFT. It does not generate test vectors (ATPG) or implement other DFT features such as MBIST, JTAG macros, boundary scan, or scan-chain compression. Implementation of these DFT features including ATPG, scan chain testing, MBIST/LBIST, JTAG macros, boundary scan, and scan-chain compression need to be performed using a separate DFT tool, such as **FAULT.**

4.2. Analog/Mixed Signal/RF

Since analog, mixed-signal, and RF designs do not depend that much on the performance of the tools but rather on the functionalities available and on the quality of the results, the approach presented in this section differs from the previous one. The analysis to be presented below is descriptive, focusing on these two aspects: the capabilities offered by the tools and the accuracy and reliability of the design outcomes.

The gap analysis will be divided into the following subsections as in previous chapters.

Analog and Mixed-Signal design

In the open-source domain, the analog design flow is largely a **manual, feed-forward process**, lacking a central management component such as a library manager or an open-access design database, which are commonly available in commercial toolchains. Consequently, the responsibility for managing multiple design views and their respective states is delegated to the user, requiring a high level of awareness and discipline from the designer.

The flow consists of a set of loosely coupled tools that are largely interchangeable, as data exchange is based on standardized file formats such as **SPICE netlists** and **GDSII** layout files. Design

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

data are typically stored in **human-readable ASCII formats**, which facilitates transparency, inspection, and, when necessary, manual manipulation of the design information.

Schematic capture tools use their own proprietary symbol formats, which are generally not interchangeable. The schematic views themselves are also not yet portable between different schematic editors, whether proprietary or open source.

Xschem provides a highly customizable and lightweight schematic editor capable of handling hierarchical designs, generating multiple netlist formats, invoking various simulation tools, and displaying simulation results within a single integrated interface. Its look and workflow are similar to **Cadence Virtuoso**, allowing users to migrate from proprietary environments to benefit from familiar key bindings and interaction paradigms.

The main challenge arises from the combination of a graphical interface with a text-based configuration environment, primarily based on **Tcl**. While this approach is powerful and flexible for experienced users, it may be confusing for newcomers. However, a comparable learning curve exists in proprietary tools, where inexperienced designers are often confronted with complex menus and extensive configuration options.

Xschem demonstrates active development and is supported by high-quality documentation. Since its early adoption as the primary schematic editor in the open-source silicon ecosystem, it has been integrated into all major open-source PDKs. Finally, the absence of a common design database and associated synchronization mechanisms prevent **back-annotation** of changes introduced at different stages of the design flow, such as layout editing or parasitic extraction.

Qucs – is an open-source electronic circuit simulator. It allows users to design and simulate analog, digital, and mixed-signal circuits using a graphical interface. QUCS includes a schematic capture tool, a simulation backend, and various analysis types such as DC, AC, transient, and S-parameter simulations. It uses its own solver (not SPICE-based) and was the predecessor to **Qucs-S**, which later added SPICE compatibility. The project is targeting Verilog-A netlist and using gnucap for simulation.

XIC Tools is a graphical editor and design suite that provides both schematic capture and mask layout capabilities. The key advantage of this toolset is its **multi-platform support**, as it runs on **Windows, macOS, and Linux** systems. Moreover, the entire design process can be carried out within a single, unified graphical user interface. Since these tools were originally developed as proprietary software and have only recently been released into the public domain, they have not yet been thoroughly evaluated by the open-source community. Nevertheless, extensive documentation is available, and the tools offer a degree of compatibility with **OpenAccess** database formats.

Revolution EDA – Revolution EDA offers symbol, schematic and layout editors for integrated circuit design. It is free to use and source-code is open to use and modify (but not to sell further). Revolution

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

EDA is rapidly developed thanks to extensive Python libraries. The project claims to release an analogue design environment that will target the Xyce simulator. Although the code is visible on the github the software is redistributed under Common Clause 1.0 license which is not open-source compatible. The software is under development.

Simulators

Since circuit simulation is one of the most critical stages of the design process, the availability of robust simulator functionality, support for industry-grade device models, and the accuracy of simulation results are of paramount importance. As summarized in the previous section, five open-source simulators collectively provide the majority of features required for analog circuit design. However, in contrast to proprietary environments, open-source toolchains generally lack higher-level abstractions for simulation setup, simulator invocation, and automated analysis of simulation results.

In recent years, open-source simulators have added support for device models written in **Verilog-A/AMS**, making model handling a critical aspect of the simulation process. Proprietary tools typically provide built-in mechanisms for parsing model cards and sweeping parameter values. The evolution of industry standards has introduced domain-specific modeling languages such as **Verilog-A** and **Verilog-AMS**, enabling designers and foundries to describe device and circuit behavior in a programmable and standardized form.

Because proprietary simulators are closed-source, they natively support these standardized languages. In contrast, open-source simulators have adopted a different approach: developers have focused on exposing built-in models to users and enabling code-based models through extensions such as **XSPICE**, or via model generators such as **OpenVAF, ADMS** and **GnucapModelGen**. These tools translate Verilog-A source code into binary data compiled and linked with the simulator.

More recently, **Ngspice** and **VACASK** introduced a standardized communication mechanism between simulators and external models through the **Open Source Device Interface (OSDI)**. VACASK is designed around the concept of implementing all device models directly in Verilog-A and relying on OSDI for simulator–model interaction. To further bridge the gap between Verilog-A and open-source simulators, an open-source Verilog-A compiler, **OPENVAF**, is currently under development.

Verilog A compliance in **OPENVAF** – according to the developers, the subset of the standard that is used by compact models (and more) is already implemented. The project delivers also some extra features not covered in Verilog-A standard.

Below is a list of **Verilog AMS** features not yet covered by gnucap-mg-gen:

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

(final_step), (absdelta), (noise_table, noise_table_log), ($display), ($monitor), ($stop), ($fatal), ($warning), ($error), ($info), ($realtime), ($stime), ($random), ($arandom), ($simparam$str), ($simprobe), ($xposition), ($yposition), ($angle), ($hflip), ($vflip) but under development.

Also, not all primitive devices are covered.

To summarize, the landscape for simulators and model support is fragmented but evolving. The comparative analysis of the simulators shows that a few of the advanced analyses are not yet supported or are being developed. Also support for statistical modeling of the circuit is something that can be done using some coding overhead. At the operational level, the simulators are very similar but not fully compatible. There are a few differences in the syntax, especially when setting up the simulations. Also, the model cards used by proprietary simulators are not identical but roughly 90% compatible with the open-source equivalents. Once the model translation is finished, the quality of results is in general very good for the available analysis. Introduction of OPENVAF Verilog-A compiler homogenizes the modeling part and permits us to use industry grade complex models. In terms of productivity, the open-source solutions lack of an integrated development environment (IDE) editor for Verilog AMS or any other mixed-signal HDL. This type of tool would help to code models more efficiently, avoid syntax errors and debug their compilation as shared libraries to be used in mixed-signal simulations.

An important difference between proprietary simulation software and open-source equivalents is the user interface. Licensed software sets up simulations and runs the simulator in the background, allowing users to view and manipulate results through the front-end interface. In the open-source domain, this process is usually much closer to the user (involving netlist operations, textual setup of simulations, and third-party tools for data post-processing). On one hand, this makes it more complex, but on the other hand, it offers greater flexibility and plenty of room for customization. A particular open-source functionality that is commonly missing in the mixed-signal design methodology is the automatic partitioning of the design between analog and digital domains, which would speed up the netlisting process. In principle, this functionality could be easily implemented by the possibility of specifying netlisting rules by the user, like cell view priorities.

Layout drawing and PyCell support

This part of the flow is quite well covered by existing tools. Recent advances in KLayout made it possible to use PyCellStudio compatible Python code to generate the layout objects. Also, the tendency to use Python language makes this step of design versatile and speed up the development process. The gap between the proprietary software is the integration with other design steps to provide data exchange and synchronization between schematic editor, EM simulator.

DRC

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

Here, similarly, on the proprietary side the generic DRC rules are implemented using tool-specific format, which is an inoptimal solution. Since DRC checks can be computationally exhaustive, the performance of DRC engine matters and depends on the implementation. In KLayout the DRC rules are typically implemented using ruby language and executed using a built-in ruby interpreter.

LVS

Open-source LVS verification shares the same problems as DRC: lack of unification and performance issues. Additionally, in some cases LVS is performed in two steps using different standalone tools: magic for netlist extraction and netgen for netlist comparison.

PEX

In the analog domain PEX extraction is still at very early stage. It is possible to extract a netlist with annotated parasitics out from the layout. However, the built-in algorithms behind these solutions are rudimentary and to not consider more complex capacitive and resistive parasitic coupling. There are other tools like FastCap, FasterCap and FastHenry, which can extract capacitance from a given geometry. The last option are the EM solvers, which use computationally intensive complex method to identify scattering parameters of an element which can lead to elements value identification. The ecosystem lacks integration at each level and the quality of results has not been verified extensively.

RF design

The part of the open-source ecosystem which relates to the RF design suffers from fragmentation and currently it is a first approximation to build a flow for RF design based on existing tools, which support to some extent the features needed to design a microwave circuit. As in the previous examples, integration is the key to solving the problem.

One common gap in using EM simulation results with open-source workflows is the use of S-parameters. Simulators typically don't support SnP data blocks (frequency domain data) for large signal/transient simulation. Convolution methods that are used in commercial circuit simulators to solve this are not yet available.

Commercial simulator frameworks often provide some "Broad Band SPICE" extraction to convert S-Parameters to a SPICE model that is valid for wideband and well behaved (passive, causal). The current state of the open-source ecosystem can support this approach.

Another pitfall is the lack, or poor quality of results, of some simulation types like phase noise, transient noise, harmonic balance, periodic steady state.

Commercial frameworks like ADS provide geometry preprocessing to make layouts more "simulator friendly" before sending them to EM. This reduces complexity without losing important details, to

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

simulate faster or make simulation possible at all. Such geometry preprocessing is certainly possible using KLayout plus possibly some external code, but this is not implemented today. The present IHP EM workflows for openEMS and Palace are based on GDSII layouts. They can be extended to multi-chip simulation, from multiple GDSII data sources, but do not support additional 3D features like bond wires, dielectric lenses or hollow waveguides.

4.3. Integrated Photonics

In PIC design, the open-source tools cover current existing needs quite well. This covers electro-magnetic device level simulations, linear circuit simulations, and layout.

However, as the domain becomes more mature, more advanced simulations and techniques will be needed: cross-talk extraction, post-layout simulation, etc., and there are currently no open-source tools dedicated to PICs.

Another problem of the ecosystem is fragmentation of the tools. There are no environments which offer seamless connection between various design steps, and manual effort is required to make sure the tools work together in a coherent design flow. Standardizing the input / output data formats could be helpful.

Taking into account large variety of tools and technologies in integrated photonics, there is a scalability problem of how foundry data propagates to software-specific PDKs. This problem is relevant for both commercial and open-source tools. In our view, this is where open-source community can create a unique approach, specifically, by defining and using open standards for the data to be provided by foundries (building block layout, performance, technology info, and design rules) and later compiled into software PDKs.

Device simulations

There are several high-quality open-source tools for EM simulation implementing various simulation methods. The main gap is that each tool provides its own method, and there are no environments where multiple methods can be employed or their results linked together. Therefore, integrating these tools together would be helpful for device design. Availability and ease of using open source PDKs can also be improved.

Circuit simulations

Linear simulations are somewhat comparable to existing commercial tools. When simulating active components, there is a lack of tools taking into account non-linear effects. This is a difficult problem, which was solved (or is being addressed) by several commercial tools, but the open-source

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

solutions are lacking in this regard. Also, capabilities to do circuit variation analysis should be improved.

Layout tools

Open-source tools enable high quality layout with many useful features such as parametric component definition, hierarchical design, high speed rendering. This requires, however, manual layout scripting and some proficiency with coding. Commercial tools allow automatic layout generation from schematic.

Some advanced features such as auto-routing are not very mature. Drag-and-drop approach and bidirectional synchronization of schematic and layout (script) are only available in commercial software tools.

Physical verification

DRC checks in KLayout are a powerful tool allowing quite sophisticated mask-level checks. The DRC execution time on large circuits (e.g. on wafer scale) with curved paths containing many points may not be optimal compared to the commercial tools. The DRC decks for a given technology are described in the dedicated tool-specific language, which therefore requires double effort when creating rule decks for different tools.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

5. Tool Recommendations

This section gives recommendations on open-source tools to be used in digital, analog, mixed signal, RF and photonics design flows. The intention is to provide at least two options for each flow. However, as each of the flow consists of multiple stages each using different tools, there are not always multiple independent open-source tools available. The tool recommendations are first described textually and then summarized in tables. The recommendations should be considered as open suggestions as the tool landscape constantly evolves, and it is important to monitor the state of tool projects closely.

5.1. Digital

System Level Design

Modern SoC designs typically consist of a programmable host core orchestrating the execution and programmable task- or domain-specific accelerators. In addition, the most energy-efficient acceleration of workloads can be achieved with fixed-function accelerators. Tools for designing these components are recommended in this subsection.

RISC-V based processors and toolsets have been adopted widely in recent years, so recommendations for them are provided. **Chipyard** is recommended for generating and implementing RISC-V based cores, as it offers six different cores as a base and allows customization. Additionally, the generated cores are tapeout-proven. The toolset utilizes generic RISC-V tools.

OpenASIP toolset is recommended for generating task- or domain-specific, compiler programmable accelerators. The toolset generates cores based on a statically scheduled wide-issue processor template whose resources and instruction set can be customized. The instruction set is automatically adapted to new operations. They are also supported by a retargetable compiler and instruction set simulator.

HLS tools are recommended for fixed-function accelerator design. **Bambu** and **Dynamatic** are recommended, as they are the most prominent open-source HLS tools. Although they are somewhat lacking in the amount of included IP libraries and extensive verification, they can produce RTL from C/C++ inputs as well as GCC/LLVM intermediate representation formats.

QEMU, **Gem5** and **Renode** are recommended for system simulation. **QEMU** is a widely adopted emulation tool with extensive support for different targets. **Gem5** is a well-established simulator with support for system-level as well as microarchitecture-level simulations. It ships with a variety of component simulation models. **Renode** aims to support platform simulations including peripherals and provide testing and verification automation functionalities. **Renode** is a functional simulator

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

and does not provide a similar level of microarchitecture modeling as **gem5**. It allows co-simulation with Verilog/SystemVerilog files.

Logic Synthesis and Physical Design (Back-End)

OpenROAD and **LibreLane** are recommended for running logic synthesis and physical design as they are the only two fully open-source toolsets for this purpose. They are currently highly active projects aiming to automatically take RTL designs as input, run physical design, and produce GDSII output. In the case of **OpenROAD**, **Openroad-flow-scripts** is recommended to be used for automating the flow. Below are limitations of the tools:

- **OpenROAD**: For VHDL support, the existing GHDL plugin should be used. Note: The plugin has limited support for VHDL. At least package files are not supported.
- **OpenROAD**: Systemverilog is supported via Slang plugin. Package files are not supported. SystemVerilog-2023 is not supported.
- **OpenROAD**: Detailed routing consumes a large amount of memory and time with large designs.
- **OpenROAD** & **LibreLane**: No DFT insertion support. **Difetto** is a work-in-progress **LibreLane** plugin for DFT insertion.

Table 5.1. Digital Design Tool Recommendations: System Level EDA Tools.

System Level EDA Tools		
Purpose/Tool Type	**Tool**	**Justification**
Simulator (Prototyping)	Gem5	Widely adopted. Wide set of component models for simulation.
	QEMU	A fast, open-source system emulator that operates at a higher abstraction level than gem5, favoring execution speed over microarchitectural detail.
	Renode	Virtual SoC simulation. Supports co-simulation.

Table 5.2. Digital Design Tool Recommendations: IP Design and Integration.

IP/HDL Design/Generation and Integration		
Purpose/Tool Type	**Tool**	**Justification**
RISC-V core design, customization, compilation and RTL generation	Chipyard	Generates tapeout-proven RISC-V cores. Contains generic *riscv-tools* for compilation, instruction set simulation etc.
ASIP design, customization, compilation and RTL generation	OpenASIP	Generates tapeout-proven statically scheduled cores. Includes customizable instruction set and retargetable instruction set simulator and compiler, and RTL generation. Supports C and OpenCL. No similar open-source ASIP toolsets are available.
Fixed-function accelerator generation	Bambu, Dynamatic	The most prominent HLS tools. Lack of other options.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

IP Integration	Kactus2	Has been used in a taped-out design. No viable alternatives.

Table 5.3. Digital Design Tool Recommendations: RTL2GDSII Flow.

RTL2GDSII Physical Implementation Flow (Logic Synthesis & Physical design)		
Purpose/Tool Type	**Tool**	**Justification**
Full Automation, Standalone	OpenROAD & ORFS	Highly active projects aiming to automatically take RTL designs as input, run physical design, and produce GDSII output. Proven by multiple tapeouts.
Automation, Custom Integration	LibreLane	Highly active projects aiming to automatically take RTL designs as input, run physical design, and produce GDSII output. Proven by multiple tapeouts.

Table 5.4. Digital Design Tool Recommendations: Synthesis, Simulation & Verification.

Synthesis, Simulation & Verification Tools			
Purpose/Tool Type	**Tool**	**Integrated/Installed with OR Framework**	**Justification**
RTL Simulation: Verilog/SV	Icarus, Verilator		Both are widely adopted. Verilator seems to be under active development.
RTL Simulation: VHDL	GHDL, nvc		GHDL is a well-established simulator and nvc is an active project with support for several verification frameworks.
Waveform viewing	GTKWave, Surfer		GTKWave is a well-established waveform viewer. Surfer aims to be highly configurable. Lack of other options.
RTL Testbench Environment	CocoTB		Not mandatory but enables the use of Python to write tests. Wide range of simulators supported.
Formal Verification	Eqy, sby	Yosys	Lack of other options.
Physical Verification: DRC, LVS	Klayout	OpenROAD/LibreLane	Default in OpenROAD. Lack of other options.
	Magic	LibreLane	Lack of other options.
	Netgen	LibreLane	Lack of other options.
Physical Verification: ERC	CVC		Lack of other options.

5.2. Analog/Mixed Signal/RF

In this chapter, the recommendations are primarily based on the **maturity of the tools** within their respective domains. Some of the tools have already been employed in real design projects using the **SKY130, GF180, and SG13G2 PDKs**, thereby demonstrating their practical capabilities. Nevertheless, additional tools are under active development; therefore, although the recommendations currently focus on two tools, the list should be regarded as **open and extensible**.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

5.2.1. Analog Design

Table 5.5. Analog Design Tool Recommendations: Schematic.

Analog design: Schematic capture and editing		
Purpose/Tool Type	Tool	Justification
Integrated design tool environment for schematic capture and simulation	Xschem	**Xschem** is a **Tcl-based**, well-established schematic editor with **multi-simulator support**. It provides symbol libraries for the **SKY130, GF180, and SG13G2 PDKs**. Within the open-source ecosystem, many analog designs developed using Xschem are actively maintained and continue to evolve. In addition, the tool closely resembles **Cadence Virtuoso** in both look and workflow, which facilitates the transition from proprietary to open-source design environments.
Integrated design tool environment for schematic capture and simulation	Qucs-S	The tool is functionally similar to **Keysight ADS** and is primarily used for **RF circuit design with discrete components**. Recent updates have added support for **PDK-based elements**, improving its compatibility with analog integrated circuit design flows. Qucs-S can invoke different simulators, such as **ngspice** and **Xyce**; however, integration of additional simulators is not straightforward. The tool remains under active development.

Table 5.6. Analog Design Tool Recommendations: Simulation.

Analog design: Simulation		
Purpose/Tool Type	Tool	Justification
Spice compatible circuit simulator	ngspice	An analysis of the table summarizing the features of open-source simulators indicates that **ngspice** is the most versatile option. Moreover, it is a well-established simulator within the open-source ecosystem and is widely used as the primary simulation engine. Ngspice supports also Verilog-A models via OSDI interface. The project is actively developed.
Analog circuit simulator based on Verilog-A	VACASK	The simulator is recommended because it **natively supports Verilog-A models via the OSDI interface**, which significantly improves simulation efficiency. Its **modular architecture**, with a clear separation between the simulation engine and device models, makes it particularly well suited to the open-source ecosystem, as models can be developed and maintained independently of the simulator core.

Table 5.7. Analog Design Tool Recommendations: Layout.

Analog design: Layout Editor		
Purpose/Tool Type	Tool	Justification
Layout editor	Klayout	**KLayout** is a well-established layout editor with support for the **SKY130, GF180, and SG13G2** technologies. Its native integration of **Python and Ruby** makes the tool highly versatile and customizable. An additional advantage is support for the **PyCellStudio API**, which enables seamless migration of proprietary parametric cells. The project is under active development.
Layout editor	MAGIC	Although **Magic VLSI** is a legacy tool, it continues to be used within the open-source ecosystem. It supports the **SKY130, GF180, and SG13G2** technologies. Its highly versatile **Tcl interface** and command-line support enable straightforward integration into automated design flows. The project remains under active development.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Since PEX analysis is at the preliminary stage in the open-source domain, there are not many tools that support this feature.

Table 5.8. Analog Design Tool Recommendations: Parasitics Extraction.

Analog design: PEX		
Purpose/Tool Type	Tool	Justification
A klayout compatible tool for parasitic extraction	Klayout-PEX	This tool acts as a **wrapper** around back-end tools that perform parasitic extraction. Its primary advantage is the integration with **KLayout**, along with support for **netlist reduction** options. Depending on the required **granularity of the PEX analysis**, it allows the user to select among different extraction tools and configurations to obtain either coarse or fine parasitic models. The tool is under development
Layout editor with PEX feature	MAGIC	**Magic** includes a built-in **parasitic extraction engine** capable of extracting **series resistance** and **shunt capacitance** from the metal stack. It can be used as a **stand-alone tool** via its **Tcl interface**.

Since physical verification is an important step in the IC design flow, it is recommended to perform it extensively using different tools, increasing redundancy.

Table 5.9. Analog Design Tool Recommendations: Physical Verification.

Analog design: Physical verification		
Purpose/Tool Type	Tool	Justification
DRC/LVS	Klayout	This is the **primary verification tool** used in the open-source domain for performing **design rule checking (DRC)** and **layout versus schematic (LVS)** validation, with rules implemented in **Ruby**. These capabilities have also been integrated into verification flows within **integrated design environments**, such as **RevolutionEDA**.
DRC/LVS	MAGIC+ netgen	**Magic**, together with **Netgen**, supports **layout versus schematic (LVS)** verification by extracting a netlist from the layout using Magic and subsequently comparing it with the reference netlist using Netgen.

5.2.2. Mixed-Signal Design

As mentioned earlier, the **mixed-signal design flow** in the open-source ecosystem remains highly fragmented and requires further integration. Nevertheless, ongoing development efforts allow for the formulation of a set of **practical recommendations**, which are outlined below.

Table 5.10. Mixed-Signal Design Tool Recommendations: Schematic.

AMS design: Schematic editing		
Purpose/Tool Type	Tool	Justification
Schematic capture	xschem	**Xschem** provides partial support for **mixed-signal simulation** through integration with **ngspice/XSPICE** and **Verilator/Icarus Verilog**. It allows these tools to be invoked and the required libraries to be linked directly from the Xschem GUI, thereby improving usability for **AMS design** and abstracting low-level command-line operations.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Table 5.11. Mixed-Signal Design Tool Recommendations: Simulation.

AMS design: Simulation		
Purpose/Tool Type	Tool	Justification
Spice compatible circuit simulator	Ngspice + XSPICE + verilator/iverilog	**Ngspice**, together with the **XSPICE code model library**, supports **mixed-signal design** using an **analog-on-top** approach. It also enables simulation of digital blocks compiled from **behavioral Verilog HDL** descriptions.
VerilogAMS and Spice circuit simulator	Gnucap	**Gnucap** is currently the only simulator that natively supports **mixed-mode simulation** by handling both **Verilog-AMS models and SPICE** within a single netlist. The project targets **Verilog-AMS 2.4 LRM** compatibility and already implements a large subset of the language features and device primitives.

Table 5.12. Mixed-Signal Design Tool Recommendations: Physical Implementation.

AMS design: Physical Implementation		
Purpose/Tool Type	Tool	Justification
Place and route tool	OpenROAD	The **OpenROAD** tool supports the use of **macros**, enabling its application to the physical implementation of **mixed-signal circuits** using a **digital-on-top** approach. In this context, the analog subsystem can be abstracted as a macro and represented using standard data formats such as **LIB, LEF, and GDS**.
Layout editor	KLayout	**KLayout** can be used in the mixed signal design at the physical implementation stage, while using **an analog-on-top** approach.

Table 5.13. Mixed-Signal Design Tool Recommendations: Macro Abstraction.

AMS: Macro abstraction		
Purpose/Tool Type	Tool	Justification
Characterization tool	lc-time	A **tool for analog circuit characterization** and for generating **Liberty (.lib) files**.
Layout editor	Magic	**Magic** is capable of writing an abstract LEF file from a GDS which is useful for macro abstraction.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

5.2.3. Radio Frequency Design

Similarly to the AMS flow, the **RF design flow** remains fragmented and lacks integration. The release of the **SG13G2 PDK** served as a significant stimulus, after which many tools improved their capabilities for **RF circuit design**.

Table 5.14. Radio Frequency Design Tool Recommendations: Schematic.

RF design: Schematic capture		
Purpose/Tool Type	**Tool**	**Justification**
Integrated design tool environment for schematic capture and simulation	Qucs-S	Since **Qucs-S** closely resembles **Keysight ADS**, a dedicated RF design environment, it is recommended as a **primary schematic editor and design tool** for RF applications. It supports the **ngspice** and **Xyce** simulators, provides **SG13G2 PDK** elements, and enables **CDL netlisting** for LVS verification.
Integrated design tool environment for schematic capture and simulation	Xschem	Since **Xschem** is well established within the **open-source IC design community**, it can also be used as a **schematic capture and design environment** for **mmWave circuits**.

Table 5.15. Radio Frequency Design Tool Recommendations: Simulator.

RF design: Simulator		
Purpose/Tool Type	**Tool**	**Justification**
Spice simulator with RF features	ngspice	Currently, it offers the **richest set of features** required for **mmWave design**, including **S-parameter simulation with noise analysis** and **Touchstone file support**.
Modern Verilog-A simulator	VACASK	A **modern Verilog-A circuit simulator** targeting **analog and RF circuit design**. Although alternatives exist, this project features a **clear roadmap** and **strong development momentum**.

Table 5.16. Radio Frequency Design Tool Recommendations: Layout.

RF design: Layout Editor		
Purpose/Tool Type	**Tool**	**Justification**
mmWave layout tool	KLayout	**KLayout** is a well-established layout editor with support for the **SKY130, GF180, and SG13G2** technologies. Its native integration of **Python and Ruby** makes the tool highly versatile and customizable. An additional advantage is support for the **PyCellStudio API**, which enables seamless migration of proprietary parametric cells. The project is under active development
mmWave layout tool	GDSFactory	**GDSFactory** was originally developed to support **photonic circuit design**. It is **Python-based**, which enables high scriptability and provides a high degree of freedom in exploring the design space. Since the geometry of individual elements plays a critical role in **mmWave circuit design**, an automated, script-driven approach can significantly aid layout optimization. **GDSFactory** is well suited to support such design scenarios.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

There are **limited options** available for **EM analysis of mmWave circuits**. Because EM simulations are **computationally intensive**, the choice of simulator depends on the **complexity of the analysis**, the **design size**, and the **available computational resources**.

Table 5.17. Radio Frequency Design Tool Recommendations: EM Simulation.

RF design: EM simulations		
Purpose/Tool Type	Tool	Justification
Electromagnetic simulation	OpenEMS	An **FDTD (finite-difference time-domain)** electromagnetic solver suitable for **circuit-level simulations**, offering a good balance between **accuracy and computational resource usage**.
Electromagnetic simulation	AWS Palace	FEM (finite elements in frequency domain) since the FEM method is more resource hungry, it is recommended for small circuits. It has more flexibility at meshing stage which translates to better accuracy for the same problem size.

5.3. Integrated Photonics

This section contains a list of various tools selected as recommended components for open-source PIC design flow. The main goal in tool selection was to cover various simulation methods and functionality. The integration between the tools is currently quite poor, but we consider the selected tools to be most suitable to support such integration in future.

The tables below contain the tools which are grouped by the design step.

Table 5.18. Integrated Photonics Design Tool Recommendations: Device-Level Simulation.

Device-level simulation		
Purpose/Tool Type	Tool	Justification
FEM simulators	**FEMWELL** **Palace**	Applicability to photonic applications, and integrations with other software.
Other simulator types	**MPB** **Meep** **Meow** **Tiny3D**	Documentation quality, applicability to photonics domain, and integrations with other software.

Table 5.19. Integrated Photonics Design Tool Recommendations: Circuit Simulation.

Circuit simulation		
Purpose/Tool Type	Tool	Justification
S-parameter circuit simulator	**SAX** **Simphony**	Good integration with other software.
Schematic editor and simulator	**Qucs-S**	The tool resembles Keysight ADS environment and was shown to be useful in photonic applications, though with some custom tweaking.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

Comprehensive photonic circuit simulator	**Phisim**	Allows amplifier simulation

There are limited options for the *layout tools* which have a scripted approach, have multiple PDKs available for them, and allow device or component abstraction level (beyond drawing polygons). The listed tools are de-facto standard in the PIC design community.

Table 5.20. Integrated Photonics Design Tool Recommendations: Layout.

Layout		
Purpose/Tool Type	**Tool**	**Justification**
Layout editor	**Nazca Design GDSFactory**	Comprehensive script-based (python) design tools. Allow creation of own component libraries. Have integrations with multiple 3rd-party tools.
Layout viewer	**KLayout**	De-facto standard viewer for PIC design. Contains PDK implementation for one of the open-source PDKs (SiEPIC).

There are limited open-source options available for physical verification. The existing tools, however, provide quite a comprehensive functionality for DRC and LVS.

Table 5.21. Integrated Photonics Design Tool Recommendations: Physical Verification.

Physical verification		
Purpose/Tool Type	**Tool**	**Justification**
DRC / LVS engine	**KLayout**	Powerful engine with own scripting language and viewing capability. Can be used in both GUI and batch mode.
DRC / LVS engine	**GDSFactory**	Integrated with design environment. Allows Python scripting of DRC / LVS.

This project has received funding from the EU DIGITAL under Grant Agreement no 101218811.

6. Conclusions

This document described open-source tool flows in the domains of digital, analog, mixed signal, radio frequency and photonics integrated circuit design. The typical flows in each domain were described, existing options for open-source tools listed, and their features compared. Then a gap analysis highlighting the most critical gaps with regards to implementing a full design flow was presented. Finally, recommendations were given for tools to be used in each of the domains.

For digital design, there exist two major frameworks, **OpenROAD** and **LibreLane** that seem to have been widely adopted by the community in recent years. Multiple tapeouts have been done with the tools and there do not seem to be major gaps in the toolflows. However, functionalities such as DFT support have been added quite recently and are not extensive.

In the analog and mixed-signal domain, the open-source design flow is functional but remains fragmented, relying on loosely coupled tools and file-based data exchange rather than a unified design database. Mature tools such as **Xschem**, **ngspice**, **KLayout**, and **Magic** enable complete schematic capture, simulation, layout, and verification flows for several open-source PDKs, including **SKY130, GF180**, and **SG13G2**. Recent advances in Verilog-A/AMS support, OSDI-based model integration, and Python-based parametric cells have significantly improved modeling consistency and portability. However, limitations remain in terms of flow integration, back-annotation, and standardized representations of technology and verification data.

In the RF and mmWave domain, open-source tooling has progressed substantially, particularly following the release of the **SG13G2** PDK, which catalyzed improved support for RF-specific components and workflows. Tools such as **Qucs-S**, **ngspice**, **VACASK**, **Klayout**, **GDSFactory** and emerging EM solvers enable RF schematic design, simulation, and, to a limited extent, electromagnetic analysis. Nevertheless, RF flows are constrained by the scarcity of mature open-source EM solvers, high computational demands, and the absence of tightly integrated simulation–layout–verification workflows. As a result, RF and mmWave design in the open-source ecosystem is feasible but still requires careful tool selection and a high level of user expertise.

In integrated photonics domain, there are open-source tools covering individual parts of the design flow, however the complete flow is fragmented. There are two main open-source layout tools, **Nazca Design** and **GDSFactory**, which are both script-based (Python) tools. Integration of these tools with circuit-level simulators is either not existing, or only available in paid version of the tools. Mask viewing and engines for physical verification are provided by **KLayout**. Circuit-level simulators are the tools which were originally developed for RF and analog domain and later adapted to photonics, which may require additional effort and expertise to make them work properly. This, and the lack of data or interface compatibility between various tools are the main pain points which make PIC design flow fragmented and difficult to set up.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.

6.1. Outlook

Future versions of this document will include a chapter with an introduction to open EDA tools relevant to an Assembly Design Flow, along with an evaluation of their capabilities and limitations. Here we will use input from APECS pilot line.

Also targets and results of coming projects from HORIZON-JU-Chips-2025-IA-EDA call on Open-source EDA tools development will be incorporated in this document.

This project has received funding from the EU
DIGITAL under Grant Agreement no 101218811.