Scalable Chip Design for Tile-Based Heterogeneous Architectures

Maico Cassel dos Santos

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2025

© 2025

Maico Cassel dos Santos

All Rights Reserved

Abstract

Scalable Chip Design for Tile-Based Heterogeneous Architectures

Maico Cassel dos Santos

System-on-Chip (SoC) design never has posed so many challenges as today. With the slowdown of Moore's Law and the end of Dennard scaling, performance gains shifted from device scaling to architectural innovations. After mid-2000s, multi-core processors supplanted single-core processors to improve performance through parallelism. Not long after, the diminishing returns in performance by adding cores marked the rise of heterogeneous SoCs. By combining general purpose cores and specialized hardware accelerators, these architectures deliver superior performance and energy efficiency compared to homogeneous SoCs, at the cost of more design complexity.

The combined escalation of SoC architectural complexity, electronic design automation (EDA) tool sophistication to handle large high-density designs, and advanced semiconductor technology process have driven steep increases in non-recurrent engineering (NRE) costs. State-of-the-art SoCs now demand investments of hundreds of millions of dollars. Compounding this issue, industry projections indicate a significant shortage of skilled semiconductor engineers in the coming years. In response, the open-source hardware (OSH) community has sought to mitigate design complexity by promoting reuse through shared component libraries, design methodologies, and software-hardware platforms.

Despite these efforts, most of the heterogeneous SoCs based on OSH that have been presented in the literature are limited to a few accelerators and general-purpose processors. This limitation stems from the complexity of designing and integrating such architectures, as well as the high

degree of customization required across architecture, process technology, and EDA tools. To foster the development of complex SoC, design methodologies must support scalability, flexibility across diverse applications, and robustness in achieving correct layouts - all with minimal NRE cost. My thesis is that *a scalable, flexible, and robust chip design methodology for domain-specific SoC has its foundation on the regularity of heterogeneous tile-based architectures, which enable computer-aided design tools to handle efficiently the complexity of chip design.*

Throughout this dissertation, I demonstrate how tile-based architectures, when combined with a tailored design methodology, facilitate the partitioning and integration of hardware components, simplify the design flow, promote automation, and significantly reduce NRE costs. The methodology builds on ESP, an open-source platform for tile-based heterogeneous SoC design, by extending its capabilities to support full chip implementation through the introduction of solutions that address all aspects of the flow, including the design hierarchy, clock- and power- domain strategies, physical implementation of individual tiles, and top-level integration.

The potential of the methodology is demonstrated through the successful tape-out of two chips: EPOCHS-0 and EPOCHS-1. Both were fabricated in a 12 nm technology, completed in under four months, and designed by a team of fewer than ten people, including researchers, PhD students, and postdoctoral fellows from multiple institutions. EPOCHS-0 (4×4 tiles) integrates seven accelerators, four RISC-V CVA6 cores, and memory tiles with off-chip interfaces. EPOCHS-1 scales to 6×6 tiles with 23 accelerators of 14 distinct types, Linux symmetric multiprocessing (SMP) support, and distributed power management - arguably the most complex SoC to date designed in an academic environment. This scalability was made possible through OSH and the reuse of community-developed accelerators. Lessons learned from these chips guided further improvements to the methodology, including a more reliable design hierarchy, automated top-level floorplanning, and a customized top-level clock tree tailored for tile-based architectures, which delivers superior scalability compared to default EDA tool configurations.

The contributions of this dissertation will remain with the System-Level Design Group at Columbia University, to realize further improvements and future ESP-based chip designs.

Table of Contents

Acknowledgments xvii

Dedication. xx

Chapter 1: Introduction. 1

1.1 Open-source Hardware Design Methodologies 7

1.2 Research Contributions 9

1.3 Dissertation Organization 11

Chapter 2: Background. 13

2.1 Digital ASIC Design Flow 14

2.1.1 The Standard Flow 15

2.1.2 Semiconductor Technology Process 20

2.1.3 EDA Tools 23

2.1.4 Design Speci cation 28

2.1.5 Intermediate Files 30

2.2 Embedded Scalable Platform (ESP) 31

Chapter 3: Scalable Methodology for Agile Chip Design 36

3.1 Enhancements to ESP for ASIC Design 36

3.2 Tile-Based Physical Design 41

3.3 Top-Level Physical Design 46

3.4 Veri cation 48

3.5 Testing 50

3.6 Results 53

Chapter 4: Methodology Validation with Two Chips 55

4.1 EPOCHS-0 56

4.2 EPOCHS-1 60

4.2.1 SoC Architecture 61

4.2.2 SoC Characterization 65

4.2.3 Data Orchestration in Heterogeneous SoCs 67

4.2.4 Distributed Hardware Power Management Architecture - BlitzCoin 70

4.2.5 Hybrid Uni ed Voltage Frequency Scaling (HUVFS) 76

4.2.6 Power Management Results 81

4.3 EPOCHS: Conclusion Remarks 89

Chapter 5: Methodology Optimization for Heterogeneous SoCs Design 92

5.1 SoC Probe 93

5.1.1 SoCProbe Architecture 94

5.1.2 Validation Flow and Infrastructure 97

5.1.3 SoC Integration 99

5.1.4 Chip Implementation 100

5.1.5 Case Studies 101

5.2 Handling Scalability and Robustness in Mesh NoCs 105

5.2.1 NoC Implementation Enhancements 105

5.2.2 Technology-driven Clock Tree Synthesis 111

Chapter 6: Automatic SoC Implementation 121

6.1 Overview of the Database for the Chip Design Methodology 122

6.2 SoC Automatic Floorplanning 126

6.2.1 Die, Tiles, and NoC Dimensioning 128

6.2.2 Floorplan-aware GPIO and Bump Placement 138

6.2.3 Floorplan- and Power-domain Aware Bump Placement 142

6.3 Push-button Chip Design Flow 149

Conclusion 153

Future Work 153

Capabilities Extensions 153

Optimization 155

Arti cial Intelligent Agents 157

Concluding Remarks 159

References 161

Appendix A: Methodology User Guide 172

A.1 Project Setup 172

A.2 Integration of ASIC Technology-Dependent Components into ESP 174

A.2.1 Memory Integration 174

A.2.2 Pad Integration 181

A.2.3 DCO Integration 187

A.3 ESP SoC Generation 189

A.3.1 HLS of ESP Integrated FFT Accelerator 190

A.3.2 SoC Con guration 190

A.3.3 SoC RTL Simulation 191

A.3.4 Clocking Strategy 193

A.4 ESP Physical Design Methodology 195

A.4.1 ESP SoC Design Database Generation 197

A.4.2 Logic Synthesis 198

A.4.3 Floorplan 199

A.4.4 Placement 200

A.4.5 Clock-Tree Synthesis 200

A.4.6 Route 201

A.4.7 Signoff 201

Appendix B: Chip Design Methodology Database. 203

B.1 Technology 204

B.1.1 Directory Structure 204

B.1.2 List of Metadata 209

B.2 ESP_EDA 222

B.2.1 Directory Structure 222

B.2.2 List of Files 224

B.2.3 ESP SoC Database Generation 226

B.3 Design 228

B.3.1 Directory Structure 228

B.3.2 List of Metadata 232

List of Figures

1.1 50 years of microprocessor trend data. [8] 2

1.2 (a) Logic synthesis [9]; (b) Placement [10]; (c) Global routing [11]; (d) Transversal illustration of standard cell routing [12]. 3

1.3 Design capability gap. [15] 5

1.4 The cost of chip design. [16] 6

2.1 Chip design core domains. 13

2.2 Chip design ow overview. 14

2.3 Synthesis and Place and Route ow. 16

2.4 Physical Veri cation ow. 17

2.5 Veri cation ow. 18

2.6 Technology les. 21

2.7 Implementation tools. 23

2.8 Veri cation tools. 26

2.9 Design speci cation les. 28

2.10 Tools' generated les. 30

2.11 ESP platform design ow. [60] 31

2.12 ESP tile architecture. [29] 32

2.13 ESP Graphical Interface. 34

3.1 Proposed methodology for agile chip development. 37

3.2 Memory integration overview. 39

3.3 Required methodology timing constraints. 42

3.4 Three approaches to top-level oorplanning. 43

3.5 Breakdown of power domains and pin assignment. 44

3.6 Symmetric power domain array for exible power cluster allocation. 46

3.7 Hierarchical timing closure ow. 47

3.8 (a) Example of a full-system bare-metal simulation for veri cation, (b) The chip-testing environment. 49

3.9 Design time breakdown. 53

3.10 Scalability between EPOCHS-0 and EPOCHS-1. 54

4.1 Swarm-based perception in autonomous vehicle with its key computation kernels. . 56

4.2 EPOCHS-0 architecture. 57

4.3 CS-GALS clocking and NoC router with asynchronous interface. 58

4.4 CS-GALS clocking and NoC router with asynchronous interface. 59

4.5 (a) V/F scaling for each tile, (b) Bene ts of of oading tasks to dedicated accelerators, (c) Bene ts for the targer swarm perception application, (d) SoC power breakdown. 60

4.6 EPOCHS-1 architecture. 62

4.7 The four main types of ESP tiles implemented in EPOCHS-1. Clockwise from top-left: IO, Accelerator, CVA6, and LLC. 63

4.8 Measured NoC congestion and performance, power, and energy for different congurations of the memory hierarchy. 68

4.9 Performance of different accelerators across different workload sizes when using various data access modes. 69

4.10 Comparison of centralized, hierarchical and decentralized power management schemes in a heterogeneous 4x4 SoC. 70

4.11 Example of successive coin exchanges in a 3x3 SoC. 71

4.12 Histograms of worst-case absolute error across all tiles over 1000 runs; blue bars are without random pairing and red bars are with random pairing enabled. 73

4.13 Left: dynamic error as a function of time for 10 individual Monte-Carlo runs (grey), an average of 1000 runs (blue), and a centralized sequential tile update (dashed red). Right, average error during convergence with 3σ error bars. 74

4.14 Number of packets exchanged and time (NoC cycles) to reach convergence (ε = 1%) as a function of the SoC dimension $n = \sqrt{\#\ \text{tiles}}$ when comparing the conventional exchange scheme and a scheme with dynamic timing enabled. 75

4.15 Annotated package and SoC/PIVR die photographs. 77

4.16 HUVFS architecture block diagram. 78

4.17 Left: LDO voltage drop-out with V_{in} swept from 0.5V to 1V. Right: TRO voltage-frequency characterization. 79

4.18 Comparison of evaluated power-enforcement techniques. 80

4.19 Transient response of UVFS during a frequency target step. 82

4.20 Transient response of PIVR and UVFS during a V_{in} target step. 82

4.21 Workload waveform comparison for four power enforcement techniques. 83

4.22 Power savings of the four power-enforcement techniques. 84

4.23 Silicon-measured response time of the proposed power-allocation scheme and annotated error during convergence. Dashed lines are the allocation target for the Viterbi and FFT accelerators. 85

4.24 Decentralized power allocation performance improvements. 86

4.25 Comparisons of execution time (left) and response time (right) of BlitzCoin (BC), BC-C and C-RR over different SoCs (top: 3x3, bottom: 4x4), power budgets, and workload types. 87

4.26 (i) HP and HE group voltage traces (ii) Top: power breakdown with single domain; Bottom: power breakdown and savings with HE and HP domains (iii) Workload DAG with HP and HE tile groupings (iv) Speedups over the baseline with a static power budget. 88

5.1 SoCProbe Microarchitecture. [75] 94

5.2 Validation ow: (a) recording simulation traces in RTL simulation in the normal mode and (b) stimulating the chip with the recorded vectors through an FPGA testbed in the test mode. 98

5.3 Implementation of a RISC-V CPU tile. The SoCProbe unit is highlighted in red. . . 100

5.4 Simpli ed view of the test vector for Test Case 2. 104

5.5 NoC path inside a tile. 106

5.6 NoC possible implementation shapes. 107

5.7 Level-shifters power domain placement strategy: EPOCHS vs proposed. 109

5.8 PDN design for tiles and NoC routers: EPOCHS vs proposed. 109

5.9 (a) CPU tile layout with level-shifters; (b) NoC router layout with I/O buffers. . . . 110

5.10 EPOCHS-1 chip CTS skew comparison (a) chip skews (b) intended skews 112

5.11 Clock tree topologies. [110] 113

5.12 Sensitivity of a connection delay to number of buffer stages. [112] 114

5.13 Proposed clock tree implementation. 118

5.14 NoC clock tree for a 4x4 SoC. 118

5.15 Effect on clock tree skews and timing closure in Mesh NoCs as design scales . . . 119

6.1 Database directory structure. 122

6.2 Technology directory structure. 123

6.3 ESP EDA directory structure. 124

6.4 Design directory structure. 125

6.5 Design database for the tiles and the noc. 125

6.6 Database interaction. 126

6.7 Router and Tile oorplanning parameters. 128

6.8 Top-level oorplanning parameters. 138

6.9 GPIO and signal bump placement. 139

6.10 NoC and ground bump placement. 143

6.11 Power domain bump placement. 145

6.12 Connection of power supply bumps to the RDL power stripes. 145

6.13 Example of a oorplan obtained automatically for a 4x4 SoC. 146

6.14 Example of a oorplan obtained automatically for a 6x6 SoC. 147

6.15 Scalable Chip Design Flow for Tile-based Heterogeneous Architectures. 150

6.16 Idea of frequency scaling in mesh Nocs. 156

6.17 Three possible oorplan implementations. 158

6.18 Area and frequency predictors. 159

A.1 ESP initial directory structure. 173

A.2 General purpose IO cell location parameter. 182

A.3 DCO architecture. 188

A.4 ESP SoC con guration graphical interface. 191

A.5 ESP clock strategy. 194

A.6 ESP technology directory structure. 196

A.7 ESP EDA directory structure 197

A.8 ESP SoC directory structure 198

B.1 Main database directories. 203

B.2 Technology directory structure. 205

B.3 EDA directory structure 223

B.4 EDA subdirectory structure 223

B.5 Design les 229

B.6 ESP SoC main les 229

List of Tables

2.1 Summary of le formats for chip design 19

4.1 Usage, con guration options, and key metrics of the ESP tiles of EPOCHS-1 and the IP blocks that they embed. 66

4.2 Design and integration approach of EPOCHS-1 IPs. 90

4.3 Comparison of key metrics across two chip generations. 90

5.1 Qualitative and quantitative summary of the seven test cases conducted with socprobe. 102

6.1 Key metrics comparison between EPOCHS-1 and a 6x6 replica SoC. 148

A.1 One read and write port. 180

A.2 One read and write and one read ports. 180

A.3 Two read and write ports. 180

A.4 Input pads. 184

A.5 Output pads. 184

A.6 Bidirectional pads. 184

A.7 DCO interface. 189

A.8 Clock-tree root sources according to clock strategy. 201

B.1 Technology sub-directory contents. 205

B.2 Technology main metadata. 209

B.3 Technology design rules metadata. 218

B.4 Technology le paths metadata. 219

B.5 Technology supply pin metadata. 220

B.6 EDA sub-directory contents. 224

B.7 EDA source le and generated directories. 226

B.8 EDA source and generated les. 227

B.9 Design subdirectory contents. 230

B.10 Design clock metadata. 232

B.11 Design corner metadata. 239

B.12 Design include directory metadata. 240

B.13 Design SoC metadata. 240

B.14 Design ow metadata. 241

B.15 Design NoC metadata. 243

B.16 Design tile metadata. 244

B.17 Design top metadata. 245

B.18 Design noc oorplan metadata. 246

B.19 Design tile oorplan metadata. 246

B.20 Design top-level oorplan metadata. 247

List of Acronyms

ACE AXI Coherency Extensions

ADC analog-to-digital converter

AI arti cial intelligence

APB Advanced Peripheral Bus

API application programming interface

ASIC application-speci c integrated circuit

AVD accelerator voltage domain

AXI Advanced eXtensible Interface

BC BlitzCoin

BC-C BlitzCoin-centralized

CAD computer-aided design

CDL circuit design language

CMP chemical-mechanical polishing

CNN convolutional neural network

CPF common power format

CPU central processing unit

C-RR centralized-round-robin

CS-GALS communication synchronous, globally asynchronous, locally synchronous

CSR control and status register

CTS clock tree synthesis

DAC digital-to-analog converter

DCO digitally-controlled oscillator

DDR double data rate

DEF design exchange format

DFM design-for-manufacture

DFT design-for-testability

DMA direct memory access

DRAM dynamic random-access memory

DRC design rule check

DUT design under test

EDA electronic design automation

EM electromigration

EMIR electromigration and voltage drop

ESD electrostatic discharge

ESP Embedded Scalable Platforms

EUT element under test

FFT Fast Fourier Transform

FIFO rst in, rst out

FinFET n eld-effect transistor

FMC FPGA mezzanine card

FSM nite-state machine

GDSII Graphic Database System II

GPIO general purpose I/O

GPU graphics processing unit

GUI graphical user interface

HDL hardware description language

HE high-ef ciency

HP high-performance

HUVFS hybrid uni ed voltage and frequency scaling

HVR hybrid voltage regulator

ICG integrated clock-gating

ILM interface logic model

IP intellectual property

IR voltage drop

IRDS International Roadmap for Devices and Systems

ISA instruction set architecture

L1 level 1

L2 level 2

LCM least common multiple

LDO low dropout

LEC logic equivalence checking

LEF Library Exchange Format

LLC last-level cache

LUT lookup table

LVS layout versus schematic

MAS manufacture and scoring

MS-CTS multi-source clock tree synthesis

MSV multi-supply voltage

NLP natural language processing

NoC network-on-chip

NPU neural processing unit

NRE non-recurrent engineering

NVDLA NVIDIA Deep Learning Accelerator

OCV on-chip variation

OSH open-source hardware

PDK process design kit

PDN power distribution network

PISO parallel-in, serial-out

PIVR package-integrated voltage regulator

PLL phase-locked loop

PLM private local memory

PM pattern matching

PNR place-and-route

PPA power, performance, and area

PSO power shut-off

PVT process, voltage, and temperature

QoR quality-of-results

RDL redistribution layer

ROM read-only memory

RTL register-transfer level

RVD regional voltage domain

SAIF switching activity interchange le

SDC Synopsys Design Constraint

SDF standard delay format

SIPO serial-in, parallel-out

SLM shared local memory

SMA SubMiniature version A

SMP symmetric multiprocessing

SoC system-on-chip

SRAM static random-access memory

SSO simultaneous switching outputs

STA static timing analysis

TCF toggle count format

TDC time-to-digital converter

TNS total negative slack

TRO tunable replica oscillator

TSV through silicon via

UART universal asynchronous receiver/transmitter

UPF uni ed power format

UVFS uni ed voltage and frequency scaling

UVM universal veri cation methodology

VCD value change dump

WNS worst negative slack

Acknowledgements

First, I would like to express my deepest gratitude to my advisor, Professor Luca Carloni, who trusted in my skills and brought me onboard of the SLD team at Columbia University. During the past six years at Columbia, I have counted on his support on many occasions, from my rst paper presentation at a circuit conference to my last. In addition, he provided me with opportunities that challenged me and helped me grow as both a researcher and a professional. His dedication to the development of his students is remarkable, as is his talent to select not only brilliant minds but also kind and collaborative students. I am very happy with what we have built together in these past years and for our accomplishments in pushing ESP to the next level. I am positive that ESP will y even higher in the years to come.

Second, I would like to thank my dissertation committee for investing their time and knowledge in my research throughout the years. I am especially grateful to Professor Martha Kim and Professor Kenneth Shepard, who have been on all of my research committees and have been keeping an eye on this work since its inception. At every opportunity, Professor Kim always gave insightful feedback to improve and guide the next steps of this dissertation. At the same time, Professor Shepard not only provided the computational resources crucial for this work but was also swift in addressing issues that were blocking its development. I am also indebted to Dr. Pradip Bose, whose leadership in bringing together researchers from different institutions made it possible to realize two complex chips that became the cornerstone of my PhD research. In addition, I am grateful to Dr. Gi-Joon Nam for showing interest in my work and bringing me onboard of IBM Research.

The lessons I learned from his mentorship in physical design became a valuable reference for the optimizations presented in this dissertation.

My PhD journey would not have been nearly as enjoyable without the SLD group at Columbia University, who were not only excellent teammates but also some of my closest friends throughout these years. I was especially lucky to have Joseph Zuckerman as my roommate and barbecue partner for the past five years. Together, we shared countless professional and personal milestones, from paper publications to miles of running in the city. I am also deeply grateful to Paolo Mantovani, Davide Giri, Luca Piccolboni, Guy Eischler, Kuan-Lin Chiu, Giuseppe Di Guglielmo, Jihye Kwon, and Martha Becker, who welcomed me when I first joined Columbia and introduced me to the social life in the city. Together with Biruk Seyoum, Erik Loscalzo, Gabriele Tombesi, Jenna Yang, Pei-Huan Tsai, Elli Alvanaki, Kevin Lee, William Baisi, Jessica Marossero, who later joined the group, and Natalie Raum and Elena Ferro, whom I also consider friends related to the SLD group, I had great moments of hard work, travels, and carpe diem in NYC. I would also like to thank several Columbia master's students, who completed important projects while working with the SLD group and were a pleasure to mentor – Lars Fischer, Zhengping Zhu, Jules Compter, and Zongwei Zhen.

Beyond Columbia, I was fortunate to take part in the EPOCHS project, which gave me the opportunity to collaborate with an outstanding team spanning multiple institutions. My special thanks to Martin Cochet and Karthik Swaminathan from IBM Research, as well as Tianyu Jia, Thierry Tambe, and Jeff Zhang, from Harvard University. Their commitment to achieving results was demonstrated through meetings at 12 AM, working on weekends, and long nights of chip testing. Beyond learning new aspects of chip design, working with such a result-driven team pushed me to give my best all the time. I would also like to thank the IBM Research physical design team for warmly welcoming me during my internship in the summer and fall of 2024: Mint Ponghiran, Hua Xiang, Lakshmi Reddy, Mihir Choudhury, Jenn Kazda, and Jinwook Jung.

Last but certainly not least, I would like to thank my family for their unwavering support. Marcio Vasconcellos is the closest I have to family in New York City and the one I can always

count on in North America. I am deeply grateful to my godparents, Joni and Guacira da Rocha, who visited me in New York only a few months after I moved, just to make sure I was doing well and who generously welcomed me into their home during the COVID-19 outbreak for longer than a guest should ever stay. My aunt Janaína Daudt and uncle Fogaça Stein who since when I was a teenager took me as a member of their crew and also gave me assistance during the pandemic times. My aunts Lizete Cassel and Ivete Cassel who gave me assistance at my young age. I am thankful to my siblings, Yara Daudt, Marcos Vinicius Cassel, and João Daudt, who have looked to me as a reference point. I would also like to thank my cousins who have always cheered for my success, especially Guilherme Cassel, who made it possible for my mother to attend my commencement, and Glauco da Rocha and Daniel Daudt, who are more like brothers and whose accomplishments inspired me to reach higher. I am especially grateful to my father, Alexandre Daudt dos Santos, who taught me values and focus to prioritize what truly matters, and to my mother, Margarete Cassel, who fought tirelessly and made countless sacri ces to ensure I received the best education possible. Finally, I honor my grandmothers: Edite Cassel, who at old age took me to school every morning when I was a child and supported me unfailingly until her passing; and Gloria Catarina Daudt dos Santos, who was the cornerstone of my education throughout my whole life and without whom de nitely I would not be where I am today.

Dedication

To my grandmothers Gloria Catarina Daudt dos Santos and Edite Cassel, and my parents Margarete Cassel and Alexandre Daudt dos Santos for all your efforts and sacri ces for my education.

Chapter 1: Introduction

Over the past six decades, the growth of integrated circuits has been driven by two intertwined scaling principles: Moore's Law and Dennard scaling. Moore's Law, rst posited by Gordon Moore in 1965, states that the number of transistors on a chip doubles approximately every two years with minimal increase in cost [1, 2]. Dennard scaling, introduced in 1974 by Robert Dennard and colleagues, argued that as transistor dimensions shrink, voltage and current would scale down proportionally, keeping power density roughly constant [3, 4]. These scaling laws allowed dramatic increases in clock frequency and performance without proportional increases in power consumption. However, by the mid-2000s, Dennard scaling began to break down due to leakage currents, voltage scaling limitations, and thermal constraints, leading to what is often called the power wall[5]. In this new era, architectural innovation, such as multi-core processors, specialized accelerators, and heterogeneous SoC architectures [6], became the primary path to performance and energy improvement rather than shrinking devices [7].

This paradigm shift is evident in the commercial evolution of SoC architectures. Single-core processors emphasized clock speed, as it increased linearly with the reduction of the transistor size, illustrated by the green line of Figure 1.1. Intel Pentium 4 frequency, for instance, ranged from 1.5 to 3.8 GHz as the process shrank from 180 to 65 nm in the time span of 6 years (2000 - 2006). In late 2000s, clock speed reached a plateau in new technology process nodes due to the power density limitation of the devices. This period marked the end of Dennard scaling and the starting of multi-core processors, led by Intel Core series and AMD Ryzen, followed by the inclusion of graphics processing unit (GPU) on the same die for graphics and parallel tasks. Such trend is captured by the black line in Figure 1.1. More recent mobile and embedded SoCs, such as Apple's A-series, Qualcomm Snapdragon, and Samsung Exynos, have gone further, integrating many heterogeneous components: processor clusters, neural processing unit (NPU), image signal

Figure 1.1: 50 years of microprocessor trend data. [8]

processors, video encoders/decoders, arti cial intelligence (AI) accelerators, and more. These chips must also meet strict energy, thermal, and cost constraints, not only raw performance. The combination of slowdown in clock speed growth after the end of Dennard scaling and the ever-increasing transistor density of Moore's Law, forced this transition.

In parallel with technology advances and architecture innovations, there has been a continuous progress of the EDA tools. Responsible for transforming a register-transfer level (RTL) hardware description into a manufacturable layout, they need to be capable of managing both technology and architectures to succeed in each stage of the application-speci c integrated circuit design ow. Although there are several tools and many engines inside each tool to account for these stages, logic synthesis, placement, routing, and functional veri cation are the core ones.

Logic synthesis tools, for instance, convert hardware description language code into logic gates and map them to a target technology using standard cell libraries. During this process, the architecture is represented as a directed acyclic graph, typically a Boolean network such as an And-Inverter Graph, enabling optimizations in both datapath and control logic. Figure 1.2 (a) gives a simple ex-

Figure 1.2: (a) Logic synthesis [9]; (b) Placement [10]; (c) Global routing [11]; (d) Transversal illustration of standard cell routing [12].

ample on how the same logic can be implemented in many different functionality equivalent ways, each offering trade-offs in power, performance, and area (PPA). Placement tools then distribute the cells across a Cartesian plane, seeking to minimize wirelength while ensuring that no cells overlap. Placement algorithms changed over the years to either keep up with the increase of design complexity or to achieve better results. Modern placement algorithms, as shown in Figure 1.2 (b), use electrostatic models in an iterative process, where cells are assigned charges of opposite polarity causing them to repel from each other. As they are moving apart, the eld reduces until there is not enough force to spread them out. Routing engines subsequently connect the placed cells, as illustrated in Figure 1.2 (c). These engines employ multiple metal layers arranged in grid-based structures, while strictly adhering to technology design rules such as minimum wire spacing, parallel length constraints, and geometric requirements for vias across layers. Figure 1.2 (d) provides

a cross-sectional view of standard cell connections within the metal stack of 9 layers. In advance technologies, the number of metal layers can reach 21 layers, with some of them connected by through silicon via (TSV) from underneath the cell. Finally, functional veri cation - primarily performed through netlist time-based simulation - applies test vectors to the design-under-test and compares its outputs against the expected results.

Most of these stages require solving NP-hard optimization problems and therefore rely on heuristics to avoid local minima and guide solutions toward near-optimal outcomes. A natural consequence of heuristic methods is non-determinism, which introduces unpredictability in the - nal results. This challenge has become increasingly pronounced in advanced technologies, where more heuristics are required to handle higher device densities and larger design scales. For example, when pushing frequency to the limit in a PULPino RISC-V core targeting 14 nm, an adjustment of only 10 MHz produced a 10% change in area after place-and-route. Such “noisy” tool behavior typically follows a Gaussian statistical distribution [13]. To meet design schedules, designers are often forced to target conservative speci cations rather than the full potential of the technology [14]. This gap between achievable technology limits and realized design outcomes was already evident in 2013, as highlighted by Figure 1.3, which is taken from the International Technology Roadmap for Semiconductors (now the International Roadmap for Devices and Systems (IRDS))

From 2001 to 2014, the IRDS design cost model served as a key reference for projecting advances in design technology for integrated circuits. These projections were based on metrics such as the number of transistors designed per engineer-month, the scaling of design productivity, and the scaling of NRE costs, which include engineer salaries, servers, tool licenses, and design schedules. The underlying assumption was that improvements in productivity would offset the increasing complexity of new technologies, thereby keeping overall design costs manageable. However, innovation has never strictly followed roadmaps. As illustrated in Figure 1.4, the combined escalation of SoC architectural complexity, EDA tool sophistication, and advanced technology process design kits (PDKs) has instead led to steep increases in both design costs and time-to-market.

Today, state-of-the-art SoCs require investments on the order of hundreds of millions of dollars.

Figure 1.3: Design capability gap. [15]

NRE costs escalate as process nodes shrink, mask sets grow more expensive and veri cation, testing, and validation tasks become more demanding. Design cycles often extend over many months, sometimes surpassing a year, and require large, multi-disciplinary teams to coordinate RTL design, veri cation, physical implementation, timing closure, and sign-off. At the same time, the shortage of skilled semiconductor engineers compounds the challenge. A 2024 report by McKinsey projected that, by 2029, the United States will face a demand for 88,000 semiconductor professionals and, with a projected supply of only 11,000–29,000, there will be a shortfall of 59,000–77,000 [17]. In this context, the development of effective design methodologies is not only valuable but essential.

By de nition, a design methodology is a structured, systematic framework that guides designers through the problem-solving process. In the chip design domain, it orchestrates the interplay of architecture, EDA tools, and technology to enable designers and companies to deliver SoCs

Figure 1.4: The cost of chip design. [16]

that meet speci cations while minimizing NRE costs. Traditional semiconductor companies often build on vendor-recommended EDA ows, customizing them into in-house methodologies optimized for their needs. A cornerstone of these industrial methodologies is design reuse, which reduces complexity and helps contain costs. In contrast, startups and academic labs often lack the nancial and human resources to develop and maintain such infrastructures, as well as the design legacy that supports large-scale reuse. Therefore, for them, the design of cutting-edge SoCs is largely prohibitive without methodologies and platforms that embrace OSH, to enable extensive design reuse, and effectively bridge EDA tool, architecture, and technology domains to reduce costs by an order of magnitude. [18].

1.1 Open-source Hardware Design Methodologies

Academic research has increasingly turned to novel architectures for emerging domains such as the Internet-of-Things, augmented and virtual reality, and autonomous systems. This progress has been strongly in uenced by the introduction of RISC-V [19], an open-source instruction instruction set architecture (ISA) that has lowered barriers to entry and fostered rapid innovation. The availability of an open ISA has catalyzed the development of open-source processors, enabling designs that range from ultra-low-power cores for IoT applications to Linux-capable SMP systems [20, 21, 22, 23, 24, 25, 26].

Building on these processors, several research methodologies and platforms have emerged to systematize SoC design and evaluation. Some emphasize con gurability through hardware generators and modular tile-based organizations [25, 26, 27, 28, 29], while others focus on rapid FPGA prototyping and end-to-end integration of accelerators and heterogeneous components [30, 29]. Collectively, these efforts provide reusable infrastructures that support full-stack experimentation with PPA trade-offs.

Complementary to advances in architectures and platforms, the open-source community has begun to address limitations in EDA. Initiatives such as OpenROAD and OpenLANE [31, 32] integrate diverse algorithms into cohesive ows, offering open-source support for tasks ranging from synthesis to design-for-testability (DFT). At a higher level of abstraction, methodology frameworks like m owgen and Hammer [33, 34, 30] further reduce cost and complexity by automating physical design ows, promoting code reuse, and enabling scalable PPA optimization. These contributions illustrate a broader shift from isolated tool development to holistic design methodologies capable of addressing the challenges of heterogeneous SoCs.

Despite the existence of such platforms and stand-alone methodologies, most of the heterogeneous SoCs using open-source hardware presented in the literature are limited to fairly simple architectures that feature a few accelerators, a general-purpose processor and, sometimes, a microcontroller for system wakeup [35, 36, 37, 38]. The reason lies on the complexity of designing and

integrating such architectures combined with the high customization degree involving designing chips and its dependencies: design architecture, target process technology, and EDA tools.

Architects need an intuitive user-friendly platform that remove several decisions not familiar to their expertise, to unleash the development of larger, aggressively PPA optimized, heterogeneous architectures that address not only a speci c component but all components needed by a domain-speci c application. In other words, to enable architects to design and test a domain-speci c SoC, all con gurations and customizations that hamper a fast chip tape-out need to be prede ned by the platform's design methodology - a scenario in which "limitation is freedom". At the same time, to handle the increasing SoC complexity the design methodology needs to be scalable, exible, and robust.

Scalability. A methodology is scalable if it can handle the growth in size and complexity of SoC designs with a sublinear growth of NRE costs - i.e., computation infrastructure, engineering effort, and design time. As the SoC complexity scales, more data must be processed by the EDA tools' algorithms and the machines running these tools need more cores and memory to reach the nal chip implementation ef ciently. Without a scalable design methodology, both computing power and design effort can grow exponentially, thus causing the design time to skyrocket.

Flexibility. A methodology is exible when it supports the smooth integration of a heterogeneous set of components from distinct development sources and can adapt to the use multiple project's target technologies and preferred EDA tools while meeting PPA requirements. Large and complex SoCs, for instance, often require multiple power and clock domains and their design requires high-end tools and advanced technologies to meet performance goals while keeping power dissipation under control. In contrast, smaller or simpler SoCs may have more relaxed performance and power constraints that can be met with mature technologies and the use of open-source EDA tools [32, 31].

Robustness.A methodology is robust when it not only achieves the target quality-of-results (QoR) metrics but also veri es functional correctness at each step of the design process, from RTL speci cation to the Graphic Database System II (GDSII) implementation. While EDA vendors

provide their own recommended flows, a variety of problems can still arise, including, but not limited to, logical or physical design issues, poor or inaccurate settings of the computer-aided design (CAD) tools, and inconsistencies with the technology models. The more a design flow can avoid these mistakes and promptly detect issues, the more robust is the methodology in its support of design reuse.

1.2 Research Contributions

My dissertation addresses the NRE cost problem involved in the realization of complex heterogeneous SoCs by introducing a scalable, flexible, and robust chip design methodology for heterogeneous architectures. The methodology leverages the underlying structural features of tiled-based architectures to unleash efficient automation, robust integration of open-source components from different developers, fine grain clock- and power- domain strategy, and parallel implementation of partitions. .

My thesis is that a scalable, flexible, and robust chip design methodology for domain-specific SoC has its foundation on the regularity of heterogeneous tile-based architectures, which enable computer-aided design tools to handle efficiently the complexity of chip design. I demonstrate my thesis through the following contributions:

1. Scalability. The methodology is scalable because it supports the design of increasingly complex SoCs with sublinear growth in NRE costs, as demonstrated by two successful tape-outs of fully open-source SoCs: EPOCHS-0 and EPOCHS-1. Both were fabricated in 12 nm, completed in under four months, and designed by a team of fewer than ten people, including researchers, PhD students, and postdoctoral fellows from multiple institutions. EPOCHS-0 (4×4 tiles) integrates seven accelerators, four RISC-V CVA6 cores, and memory tiles with off-chip interfaces, while EPOCHS-1 scales to 6×6 tiles with 23 accelerators of 14 distinct types, Linux SMP support, and distributed power management - arguably the most complex SoC to date designed in an academic environment. This level of scalability was enabled by

OSH and the reuse of community-developed accelerators. Complementary optimizations, including a tailored clock tree synthesis and automated oorplanning validated through multiple tape-ins, further sustain performance and keep design cycle growth under control as system size increases.

2. Flexibility. The methodology is exible because it supports the integration of new technologies and EDA tools through a structured database and user-friendly ows for technology-dependent components such as memories, general purpose I/Os (GPIOs), and digitally-controlled oscillators (DCOs). It enables tile relocation or replacement at any stage and transparent integration of accelerators from diverse sources and design ows, enabling the realization of SoCs of varying sizes and application domains. This exibility allows new architectural features to be added with low impact on downstream stages. Its effectiveness was demonstrated through EPOCHS-0 and EPOCHS-1, which include solely open-source hardware, required integration of new technology and EDA tools from the ground up, features a customized distributed power management algorithm, and successfully handled last-minute tile replacements only days before tape-out.

3. Robustness. The methodology is robust because it embeds veri cation throughout the design ow, including functional veri cation of the SoC integration, multi-stage logic equivalence checking, and FPGA emulation for larger and complex tests, to ensure correctness across all stages. It enables small teams to verify large SoCs. It partitions workloads strategically to balance complexity and mitigate risks in multi-institutional collaborations. This robustness has been showcased in both chip designs: EPOCHS-0, which booted Linux within four weeks of fabrication, and EPOCHS-1, which achieved Linux SMP boot in only three days - demonstrating reliability even in highly complex designs.

While tile-based architectures may not achieve strict Pareto-optimality in the PPA space, their regularity reduces NRE costs and accelerates design cycles, making them highly competitive in the combined PPA–NRE landscape - much like the adoption of standard cells once transformed

productivity and scalability in chip design. As the methodology evolves, steps with signi cant PPA impact can be re ned. Furthermore, with the growing role of AI-driven optimization, the remaining PPA gap can be progressively narrowed, reinforcing tile-based design as a scalable and forward-looking approach for future SoCs.

1.3 Dissertation Organization

This dissertation is organized into seven chapters that represent the chronological development of our methodology.

Chapter 2 introduces the landscape of modern chip design, tracing the ow from high-level speci cation to fabrication and highlighting the interplay of design, technology, and EDA tools. It then turns to Embedded Scalable Platforms (ESP), the open-source platform extended in this work from FPGA prototyping to full tape-out, establishing the foundation for the contributions that follow.

Chapter 3 presents the central contribution: a exible, scalable, and robust methodology for heterogeneous SoC design [39]. Building on ESP, it adds support for technology-dependent components, extends the ow to tape-out, seamlessly integrates of power and timing constraints, and strengthens veri cation across the design and test process. These advances point toward a “push-button” methodology that lowers barriers to chip design.

The demonstration of the methodology is provided in Chapter 4 through two 12 nm tape-outs: EPOCHS-0 [40], a 4×4 tile-based SoC for swarm perception integrating RISC-V CVA6 cores with signal-processing and deep-learning accelerators; and EPOCHS-1 [41], a 6×6 tile-based SoC with 23 accelerators of 14 types, featuring distributed power management and runtime orchestration of data.

Chapter 5 re ects on lessons from the EPOCHS chips, introducing three optimizations: enhanced test coverage, re nements to ESP's design hierarchy for robustness and performance, and scalability improvements to the interconnect.

Chapter 6 demonstrates ESP's potential for a fully automated RTL-to-GDSII ow. A central

database supports multiple technology nodes and EDA tools, while automation of top-level oor-planning - including network-on-chip (NoC) routers, tiles, die dimensions, and power-aware I/O - delivers a correct-by-construction, push-button ow for ESP users and developers.

Finally, Conclusion outlines directions for future research, from traditional optimization strategies to AI-driven agents that complement conventional tools. Such extensions must preserve the methodology's core principles - exibility, robustness, and scalability - while opening new avenues for innovation.

Chapter 2: Background

Chip design is a multidisciplinary work ow that integrates three core domains: process technology, EDA tools, and hardware design architecture. These domains, illustrated in Fig. 2.1, converge in an interactive RTL-to-GDSII ow that transforms high-level behavioral descriptions into a manufacturable physical layout. Although implementation methodologies differ - spanning open-source frameworks (e.g. OpenROAD [31], Hammer [42], Chipyard [30], OpenPiton [28], ESP [29]) to proprietary ows used in industry - they all conform to a standardized sequence of steps constrained by foundry design rules, tool capabilities, and architectural intent.

Figure 2.1: Chip design core domains.

This chapter does not attempt to exhaustively detail the entire design ow. Instead, it presents a high-level overview to establish foundational context for the chapters that follow. The next section outlines how these domains interact to form the standard digital application-speci c integrated circuit (ASIC) design ow while the subsequent section presents an open-source platform on which the chip methodology proposed in this work is built.

2.1 Digital ASIC Design Flow

The transformation of a RTL hardware description into a GDSII layout le suitable for fabrication is a multi-stage process involving specialized EDA tools, standardized le formats, and tightly de ned design speci cations. This RTL-to-GDSII ow spans various levels of abstraction, from behavioral descriptions to physical implementation, and relies on interoperability between formats such as Verilog, Liberty (.lib), Library Exchange Format (LEF), and Synopsys Design Constraint (SDC). Each format conveys key design data, including functionality, timing constraints, and physical characteristics. Figure 2.2 illustrates how the three foundational domains interact throughout this process.

Figure 2.2: Chip design ow overview.

The ow begins with the de nition of the hardware architecture in formats compatible with simulation and logic synthesis tools. As the design progresses, its abstraction level systematically decreases through successive stages — synthesis, placement and routing, and signoff — until the nal layout is produced. Each stage is supported by a corresponding EDA tool and appropriate abstraction of the underlying technology data, often derived from the foundry's PDK. Figure 2.2

also contrasts chip-level and standard cell development. While chip design flows from high-level functional descriptions toward low-level physical representations, standard cell development follows the inverse path, starting from detailed transistor-level layouts and producing abstracted views (e.g., .lib, LEF) consumed by higher-level design tools.

The remaining of this section expands over the standardize digital ASIC design flow, showing the main files and stages, and details key elements of each core domain.

2.1.1 The Standard Flow

The chip design flow can be organized into multiple categories and subcategories. For clarity and simplicity, this document divides the process into two primary categories: Implementation and Verification.

The Implementation category includes all design stages required to generate a final layout suitable for fabrication. These stages include RTL development — either manually written by hardware engineers or generated from high-level language subsets such as SystemC or C++ — followed by synthesis, place-and-route (PNR), and physical verification to ensure compliance with manufacturing design rules. This work builds upon an open-source platform, introduced in the next section, leveraging its RTL development methodology while extending its capabilities for rapid chip prototyping.

Figure 2.3 shows the synthesis and PNR stages within the Implementation flow, detailing their respective inputs, outputs, and dependencies, as well as their interactions with other stages in the overall design process. Each domain is represented by colors. The technology domain is represented by yellow, distinguishing whether the source files are related to the foundry – dark yellow – or the library vendor – light yellow. The blue color represents the design domain and the red the EDA tool domain. Lastly, in grey are output files resulted from the interaction of these three domains. Unless specified otherwise, this color representation is used to specify the domains in the flow across all this document.

During synthesis, the RTL is optimized and mapped to the target standard cell library to pro-

Figure 2.3: Synthesis and Place and Route ow.

duce a gate-level netlist. This netlist is then processed by the PNR stage, which places the cells and routes the interconnects according to timing, congestion, and physical design constraints.

Physical veri cation, shown in Figure 2.4 , is the last step of the implementation. It ensures the design generated meets all foundry's design rules, including design rule check (DRC), design-for-manufacture (DFM), and layout versus schematic (LVS). The design is ready for fabrication once it successfully passes through these veri cations.

While the primary objective of the implementation category is to produce a design suitable for

Figure 2.4: Physical Veri cation ow.

fabrication by the foundry, the Veri cation category ensures that the design maintains functional correctness throughout the various implementation stages. As illustrated in Figure 2.5, veri cation is typically divided into three major stages: RTL veri cation, gate-level veri cation, and signoff veri cation.

RTL veri cation is responsible for validating the functional correctness of the design at the behavioral (RTL) level. This stage also plays a critical role in generating switching activity data, which is essential for early power estimation and for guiding power-aware synthesis optimizations. Once functional correctness is con rmed, the design can be mapped to the target technology during the synthesis stage.

Gate-level veri cation occurs at multiple points within the implementation ow. In addition to functional and formal veri cation of the synthesized netlist, this stage includes performance assessments through static timing analysis (STA) and power analysis. Furthermore, logic equivalence checking is employed to formally verify that the synthesized netlist is functionally equivalent

Figure 2.5: Veri cation ow.

to the validated RTL. This approach offers greater ef ciency and reliability compared to traditional simulation-based methods.

Signoff veri cation covers many of the same analyses performed during gate-level veri cation but incorporating post-layout data to achieve higher accuracy. Operating on the detailed netlist, including parasitic extraction and realistic interconnect models, this stage performs high- delity STA and power analysis, typically with signi cantly increased computational runtime. Additionally, signoff includes rail analysis and thermal analysis. Rail analysis evaluates the integrity of the power distribution network (PDN), checking for violations related to electromigration and voltage drop (EMIR). Thermal analysis identi es potential thermal hotspots that could compromise reliability or performance.

As shown in the standard ow gures, chip design relies on several les of different formats and domains. Table 2.1 summarizes all le formats we are detailing in the following Subsections.

Table 2.1: Summary of le formats for chip design

Domain	File / Format	Purpose	Extension
Tech	GDSII	Complete physical layout exchange	.gds2
Tech	LEF	Abstract layout, standard cell library data	.lef
Tech	SPICE	Device and standard cell models for analog/mixed-signal simulation	.sp
Tech	Parasitic Extraction	Extracted resistances/capacitances for timing and SI	Tool-speci c
Tech	DRC	Manufacturability veri cation rules	.drc
Tech	LVS	Layout-to-schematic equivalence veri cation	.lvs
Tech	Pattern Matching	Lithography hotspot rules	Tool-speci c
Tech	MAS	Yield and manufacturability scoring	.mas
Tech	CDL	Netlist for LVS	.cdl
Tech	Liberty	Standard cell functional, timing, power, noise characteristics	.lib
Tech	Verilog	Logic functionality description	.v / .sv
Tech	AOCV	Process-induced delay variation	.aocv
Design	SystemC / C++	High-level hardware modeling and co design	.cpp / .h / .sc
Design	RTL (Verilog, VHDL)	Microarchitecture description	.v / .sv / .vhd
Design	Testbenches & Assertions	Veri cation environment and properties	.v / .sv
Design	Timing Constraints	Rules for correct operation and perfor mance	.sdc
Design	Power Constraints	Multi-voltage domains, power shut-offs, isolation cells	.upf
Design	Chip Floorplan	Die size, macro placement, PDN parame ters	Tool-speci c
Interm.	SDF	Gate-level timing info for simulation	.sdf
Interm.	Stimulus Files	Switching activity for power estimation	.tcf / .saif / .vcd
Interm.	DEF	Physical layout data for downstream ows	.def
Interm.	ILM	Macro interface info for STA	.ilm
Interm.	Power Pads Location	Power/ground pin and bump locations	Tool-speci c

2.1.2 Semiconductor Technology Process

The technology domain refers to which foundry and gate length process – 65 nm, 12 nm, 7 nm, etc – the integrated circuit will be fabricated. PDK) has a set of standard le formats provided by foundries (Figure 2.6 (a)) that enable analog and radio frequency (RF) designers to design circuits for speci c applications such as baseband transmitter and receivers for wireless communication, energy harvesting for chip power supply, analog-to-digital converter (ADC) and digital-to-analog converter (DAC), or standard cells. The PDK also contain les that specify the set of rules the design – digital, analog, and RF – has to meet for a successful fabrication.

In contrast to a full-custom design ow, where transistor dimensions, SPICE simulations, and the layout of circuits are custom made, digital design ow is based on pre-designed cells – called standard cells. Logic gates such as ANDs, ORs, Flip Flops, are examples of standard cells. The standard cell library is compliant with the foundry technology design rules and is usually provided by third-party vendors through licensing. To optimize runtime and computer capacity, digital EDA tools use additional le formats, also called views, that models the standard-cells in a higher level of abstraction (Figure 2.6 (c)). These views are generated through characterization tools (Figure 2.6 (b)) that extract information from the layout and SPICE simulations.

The next pages detail the les presented in Figure 2.6.

Graphic Data System II (.gds2)[43, 44] is a binary le format that describes the complete physical layout of a design, including standard cells and chip-level geometry. It is the industry-standard format for exchanging design data between different EDA tools and chip foundries for manufacturing.

Library Exchange Format (.lef) [45] is an abstract representation of the layout used to simplify physical design. Foundry-provided LEFs de nes the technology layer map (metal layers, names, widths, pitches, vias, etc.), while library vendor-provided LEFs describes the standard cell layouts without internal details presented in GDSII format. Common data included in this format are: cell dimensions, pin locations and layers, diffusion regions, and obstruction information. Logic Synthesis and PNR tools use LEF instead of GDSII for computational ef ciency.

Figure 2.6: Technology files.

SPICE (.sp) [46] it is a circuit description format used with the Simulation Program with Integrated Circuit Emphasis for accurate analog and mixed-signal simulation. Foundry SPICE files define device SPICE models, such as transistors, while libray vendors provide SPICE models for standard cells, enabling schematic-level and pre-layout simulations.

Parasitic Extraction (PEX) [47] a binary file that specifies extracted parasitic elements (resistance and capacitance) of the technology devices and interconnects. PEX data is tool- and corner-dependent with separate files generated for each process, voltage, and temperature (PVT) condition and tool vendor. File names and extensions vary across foundries and EDA tools.

Design Rule Check (.drc)[48, 47] specifies the technology rule deck used for performing DRC verification. These rules define layout constraints (e.g., spacing, width, enclosure) to ensure manufacturability. The format is tool-dependent, with one or more files provided per DRC tool vendor.

Layout Versus Schematic (.lvs)[48, 47] specifies the technology rule deck for performing LVS verification. Like DRC decks, these files are tool-dependent and consist of multiple files.

Pattern Matching (PM) [46, 47] specifies a supplementary rule deck that defines hotspot patterns identified from foundry silicon data. They are tool-dependent, with formats and extensions

varying across foundries and vendors.

Manufacturing And Scoring (.mas) [46, 47] speci es recommended design practices for yield optimization and scoring during manufacturability analysis. MAS decks are tool-dependent, may include multiple les, and are used in conjunction with DRC/PM veri cation ows.

Circuit Design Language (.cdl) [44] is the netlist standard format used primarily for LVS. It resembles SPICE and is automatically generated from schematics or HDL description (e.g., Verilog) with power and ground information.

Liberty (.lib) [46, 47] is a standard format describing the functional behavior, timing, power, and noise characteristics of standard cells. Generated by library characterization tools from SPICE models, it contains lookup tables indexed by input transition times and output capacitances. These tables capture cell delay, output slew, and power across speci c PVT conditions. Liberty les are essential for logic synthesis, STA, and PNR, offering computational ef ciency compared to SPICE simulation.

Advanced On-Chip Variation (.aocv) [49] speci es delay variations in standard cells caused by manufacturing process variability. It consists of lookup tables with axes representing spacial distance on silicon and path depth, and entries representing delay adjustments. It enables more accurate timing analysis compared to simple derating factors.

Verilog [50] is a hardware description language (HDL) used to describe the logic functionality of the standard cell and digital circuits. It supports both RTL descriptions for design and gate-level descriptions for simulation and veri cations.

The standard cell-based design ow enable design scalability and automation through reuse of pre-characterized standard cells, trading off some PPA compared to full-custom design. A common strategy for building complex, high-performance chips is to leverage the scalability and automation of digital ow for most logic while applying full-custom techniques to critical path modules.

Figure 2.7: Implementation tools.

2.1.3 EDA Tools

The standardization of le formats combined with the inherit regularity of standard cell libraries enable both the automation of most steps in the digital design ow and the scalability of SoCs.

Nowadays, EDA tools combine a subset of algorithms to complete broader task. For instance, synthesis tool use graph optimization algorithms to improve area, power, and timing of the netlist before proceeding to a technology mapping algorithm. PNR tool englobes placement, clock tree synthesis, and routing algorithms in addition to timing, power, and design rules analysis.

Figure 2.7 illustrates the major steps executed by implementation tools execute and the following paragraphs gives a broader overview of their goals.

High Level Synthesis (HLS)transforms C++ and SystemC code into HDL) such as Verilog or VHDL. HLS tools facilitate design space exploration by automatically modifying the microarchitecture and processing scheduling through simple code or in the tool-based directives. Nonetheless, traditional languages like Verilog and System Verilog are still predominant in the industry due to their ne-grained control over the microarchitecture, which allows superior PPA) optimization compared to HLS-generated designs.

Logic Synthesisperforms several optimizations on the datapath and control logic, particularly

on arithmetic units like adders and multipliers and nite-state machine (FSM), repectively. It also merges or eliminates redundant or unused registers. After optimizations, the tool maps the generic netlist to the target technology library. Further re nements include resizing standard cells or swapping them with cells of different voltage thresholds to enhance PPA.

Floorplan de nes the die size and places large amount of design components within the core area, including hard macros (e.g., memories and analog blocks) and design instances in a partitioned manner. This step also includes the creation of the PDN, which is crucial for proper chip operation, as excessive voltage drops across power stripes can cause standard cells malfunction. In addition, the top-level oorplanning places the GPIO cells, which function as level shifters between the chip core operating voltage and the board operating voltage with electrostatic discharge (ESD) protection. Because automated tools typically produce suboptimal oorplans, this is a manual and time-intensive process. A poor oorplan degrades PPA and, in extreme cases, renders a chip unfeasible.

Placementarranges the standard cells within the core area and is typically performed automatically by EDA tools, producing acceptable QoR. However, for designs with stringent performance requirements, engineers can ne-tune the tool or manually guide placement, particularly of cells on time-critical paths.

Clock Tree Synthesis (CTS)generates the clock distribution network throughout the chip. Engineers can de ne constraints for metal layer usage, net widths of clock tree structure (e.g. trunk and leaf), and other routing rules. To improve signal integrity, shielding techniques may be employed, either by tying adjacent routing tracks to power or ground, or by leaving them unrouted. Given the critical role of clock networks in synchronous systems, CTS is closely supervised by experienced engineers to ensure timing integrity and reliability.

Routing connects all placed standard cells and macros with metal wires while preserving the integrity of the clock tree. Beyond minimizing wire length, the routing engine must comply with all design rules of the target technology. Tools typically apply iterative algorithms with different objectives: rst meeting setup timing constraints, then meeting hold timing, and nally achieving

a DRC-clean layout. With the advance of technology over the years, the number of metal layers for standard cells connections increased from 3 to over 14.

Physical Veri cation is the nal step in the PNR ow, ensuring the design complies with all technology-speci c design rules. This includes checks for antenna effects, connectivity correctness, and DRC compliance. Passing physical veri cation is essential before tape-out.

Layout Versus Schematic (LVS) is a veri cation step that ensures the nal layout of the chip is functionally equivalent to its original schematic or netlist containing power and ground connections. Both the Verilog and the GDSII les are generated from PNR tools and read into layout tools, where the Verilog netlist is converted into CDL netlist format. The tool compares connectivity and component matching between the extracted layout netlist and the source netlist. A clean LVS result con rms that functional discrepancies were not introduced during the physical implementation. Failing LVS can result from missing connections, incorrect device parameters, or unintended shorts or opens, and must be resolved before tapeout.

Design Rule Checking (DRC) validates that the physical layout adheres to all geometrical and spacing rules de ned by the foundry. These rules are critical to ensure manufacturability and yield. Although routing tools aim to produce a DRC-clean layout, veri ed in the physical veri cation step, signoff DRC uses layout tools with all the information of the design layout – GDSII views – and the complete deck of rules of the technology not all included in the LEF les.

Pattern Matching (PM) is a rule-based veri cation technique that detects known problematic layout patterns that are dif cult to manufacture reliably, even if they pass standard DRC. Foundries provide libraries of forbidden or weak patterns based on previous silicon failures or yield issues. This step is particularly important at advanced nodes, where lithography limitations make certain layout geometries highly sensitive to manufacturing variations.

Manufacturing Analysis and Scoring (MAS) evaluates the robustness of the layout with respect to process variability, yield, and lithographic limitations. This includes checks of process hotspots, critical area analysis (CAA), and lithographic printability. Foundries or EDA tools may assign a Manufacturability Score of the design based on these analyses, re ecting how likely the

Figure 2.8: Veri cation tools.

chip is to yield successfully in high-volume production. A higher score correlates with greater yield robustness.

Metal Fill is the process of adding non-functional metal shapes into the layout to meet uniform metal density requirements. These lls help ensure uniform chemical-mechanical polishing (CMP) during fabrication and prevent issues like dishing or erosion.

As mentioned in previous sub-section, implementation tools alone can only assure the design manufacture feasibility. They don't verify whether the design is functionally correct. That is the goal of veri cation tools, shown in Figure 2.8.

It is possible to distinguish veri cation tools based on their primary purpose: verifying behavioral functionality or ensuring performance and physical ef ciency. The former validates the functional correctness of the design, while the latter evaluates whether the design meets performance, power, and thermal speci cations. The following paragraphs provide a high-level overview of each veri cation performed by different EDA tools.

Functional Veri cation veri es the logical correctness of the hardware design, typically written in HDL, using time-based simulation tools. It spans the entire design ow and often continues even after the chip tapeout. Functional veri cation is the most resource-intensive phase of chip development, frequently requiring the largest number of engineers in the design team.

Formal Veri cation mathematically proves that a design meets speci c properties or constraints, without relying on simulation. It complements functional veri cation by targeting edge cases or complex scenarios that would be dif cult or time-consuming to cover with simulation alone. Formal techniques are especially valuable for control logic, safety checks, and protocol

compliance.

Logic Equivalence Checking (LEC)compares the synthesized gate-level netlist with the original RTL to ensure they are logically equivalent. It is often used to con rm the that sythesis and place-and-route transormations have not altered the design's intended functionality.

Power Implementation Veri cation checks that power management structures, such as level-shifters, power switches, state-retentions, always-on cells, and isolation cells to mention but a few, are correctly implemented in the netlist. It also veri es the completeness and correctness of the power intent speci cation, usually described in uni ed power format (UPF) [51] or common power format (CPF).

Static Timing Analysis (STA) veri es that signal propagation delays meet design timing constraints without requiring simulation. The most critical checks are setup check, which ensures data arrive in time to be latched correctly, determining the maximum operating frequency of the design, and hold check, which ensures data remain stable long enough to be correctly latched, preventing race conditions. STA also analyzes signal transition times and path delays to ensure robust timing closure.

Power Analysis estimates the chip power consumption caused by both switching activity in the design (dynamic power) and the transistor's inherent leakage current (static power). Accurate power estimation is essential to meet power budgets, thermal constraints, and battery life targets in mobile applications.

Thermal Analysis evaluates heat dissipation across the chip, identifying hotspots in regions with high switching activity or gate density. Effective thermal analysis avoids performance degradation or physical damage due to excessive temperatures.

Rail Analysis (EMIR) veri es the integrity of the PDN by analyzing electromigration (EM) and the voltage drop (IR) caused by the current and the rails resistance. These checks ensure that the PDN can deliver suf cient voltage to all parts of the chip under worst case conditions.

To conclude, the three major EDA vendors – Cadence, Synopsys, and Mentor (now part of Siemens EDA) – continue to advance their veri cation and implementation tools, introducing new

Figure 2.9: Design speci cation les.

algorithms, features, and le formats to support evolving technology nodes and design complexities. What's more, most of the le formats detailed in the next pages were developed by one of these three major companies. In parallel, the academic and open-source communities remain active in researching novel methodologies and tools to improve QoR across the entire digital design ow.

2.1.4 Design Speci cation

The design domain translates product speci cations into le formats compatible with EDA tools. For instance, the functionality speci cations are capture in HDL les, performance and power consumption requirements are expressed as timing and power constraint les, and the operational environment is de ned through PVT corners.

An overview of key design les is shown in Figure 2.9 and the following paragraphs describe each in detail.

SystemC/C++[52] is a high-level coding framework used to describe intended hardware operation. SystemC enables software-hardware co-design and design space exploration, making it especially useful in early architecture exploration and the development of hardware accelerators. Its adoption has grown signi cantly in recent years for specialized and domain-speci c designs.

RTL describes the hardware microarchitecture. RTL serves as input of veri cation tools, the output of HLSHLS tools, and both the inputs and outputs of logic synthesis and PNR tools ows.

Verilog [50], SystemVerilog [53] and VHDL [54] hardware description languages are commonly used across both academia and industry.

Testbenches/Assertions define the verification environment of the design under test (DUT). Testbenches generate stimulus, captures DUT outputs, and compare them against expected outputs. SystemVerilog is the industry preferred language verification, enabling the development of universal verification methodology (UVM) [55] environments. Assertions, also written in SystemVerilog, specify formal properties and are applied in both formal and simulation-based verification tools.

Timing Constraints specify a set of timing rules that ensure the design meets functional and performance requirements. Timing constraints typically fall into five categories: operating conditions, environmental, timing, power, and design rules. Operating conditions specify the PVT corners for field operation. Environmental refers to input transition, driving cells, output load capacitance, and fanout. While timing specifies all clock-related constraints, such as domains, period, duty cycle, jitter, latency, and external delays, power constraints specify limits on dynamic and leakage power. Lastly, the design rules specify the maximum capacitance, fanout, and transition times. Timing constraints are typically expressed in SDC [56] format and are essential for both synthesis and STA.

Power Constraints specify the chip power architecture, including multi-voltage domains, power shut-offs regions, and their required infrastructure cells such as power switches, isolation cells, and level shifters. These constraints are most commonly expressed in the UPFs [51], though some vendors support proprietary formats.

Chip floorplan defines the die size, aspect ratio, placement of functional blocks within the core area. It also specifies GPIO and bump locations, along with power PDN parameters such as strap widths, spacing, and topology. A well-defined floorplan provides the foundation for routing, timing closure, and power integrity.

Figure 2.10: Tools' generated les.

2.1.5 Intermediate Files

Lastly, during the design ow, each EDA tool produces at least one output le used for analysis, downstream steps, or back-annotation to upstream tools. Some output formats may overlap with previously described les (e.g., LEF, GDSII, Liberty) but are generated at different stages; these are not repeated here to avoid redundancy. Figure 2.10 provides a complete overview of the main intermediate les.

Standard Delay Format (SDF)[57] speci es the design's delay and timing information. SDF is primarily used in gate-level simulation to accurately model timing behavior, including path delays, setup and hold times, and interconnect effects.

Stimulus File captures the switching activity of nets and gate terminals for power analysis. Stimulus can be expressed as an average of switching activity in a simulation, such as toggle count format (TCF) and switching activity interchange le (SAIF), or a time-step sample in the simulation, such as value change dump (VCD) [50]. While the former formats are useful to estimate the average power consumption for a given simulation, the latter is useful to estimate the peak power consumption in a simulation interval.

Design Exchange Format (DEF)[45] describes the physical layout of the design, including netlist, placement of standard cells and macros, and routing information. The DEF les serve as the main interface between synthesis and PNR tools.

Interface Logic Model (ILM) speci es macro-level interfaces information in hierarchical or partitioned designs, including signals from macro inputs to the rst registers and from the last

Figure 2.11: ESP platform design ow. [60]

registers to outputs. ILM les enable accurate timing analysis and static timing closure in top-level integration.

Power Pads Locationdetails the location of power and ground pins for internal macros and bumps for top-level in ip-chip implementation. This information is critical for EM analysis, IR drop estimation, and overall PDN integrity.

2.2 Embedded Scalable Platform (ESP)

ESP is a platform for heterogeneous SoC design and programming [29]. Developed over the course of more than a decade of research and teaching at Columbia University [58, 59], ESP combines a scalable architecture and a exible system-level design methodology for rapid FPGA prototyping (Figure 2.11).

The ESP architecture is structured as a heterogeneous tile grid build upon a 2D mesh, multi-plane NoC [61]. Each type of tile serves a different purpose in the SoC, but every tile is encapsulated in a modular socket. The socket decouples the design of the tile from the NoC following the principles of communication-based system-level design [62]. It also provides several platform

Figure 2.12: ESP tile architecture. [29]

services to the intellectual property (IP) block of each given tile (e.g. dynamic voltage frequency scaling, performance counters, coherence, direct memory access (DMA), etc.). The ESP architecture hence strikes a balance between *regularity* and *specialization*. Currently, there are five main types of tiles in the ESP architecture, as shown in Figure 2.12.

The *Processor Tile* contains a central processing unit (CPU) developed by a third-party vendor, instantiated off-the-shelf with its own private level 1 (L1) cache. The processor options include the 32-bit SPARC Leon3 core [63], the 32-bit RISC-V Ibex core [22], and the 64-bit RISC-V CVA6 (formerly known as Ariane) core [23]. In the case of the Leon3 and CVA6 cores, the processor tile can instantiate the ESP level 2 (L2) cache, which allows the core to transparently participate in the ESP coherence protocol, supporting multicore execution and booting Linux SMP [64, 65]. The development of the proposed methodology was driven by the design of chips that use the CVA6 core, but it could be easily extended to support also the other cores.

The *Memory Tile* provides a channel to external memory. ESP allows for the seamless instantiation of multiple memory tiles to satisfy the bandwidth requirements of large SoC designs; in such case, each memory tile serves a discrete partition of the global address space and the logic

to route requests to the appropriate tile is automatically generated. When the ESP cache hierarchy is enabled, the memory tile contains the ESP last-level cache (LLC). The LLC, together with the L2, implements a standard directory-based MESI coherence protocol, adapted to work over a NoC [64]. The ESP LLC additionally can handle DMA requests directly from accelerators in an LLC-coherent manner, as proposed in [66].

The *Auxiliary Tile* hosts the non-memory I/O interfaces of the SoC, such as Ethernet and universal asynchronous receiver/transmitter (UART), as well as miscellaneous components, such as the bootROM and interrupt controller. The Ethernet connection supports remote connection through SSH and enables the *ESPLink* debug application, which is an important part of the testing framework.

The *Shared Local Memory (SLM) Tile* can be instantiated to add to the on-chip memory capacity of SoCs. It provides a software-managed scratchpad that lies outside of the coherent address space and can be shared by multiple CPUs and accelerators that access it via DMA. The SLM tile is especially important in ASIC prototypes that may not contain a DDR controller and thus pay an increased penalty for accessing external memory.

As ESP embraces heterogeneity in SoC design, the *Accelerator Tile* is a key component of the architecture. In ESP, accelerators are given equal importance as processors in the SoC and hence occupy their own tile. They are *loosely-coupled* accelerators [66], executing coarse-grained tasks when invoked by a processor core through a device driver. When communicating with the memory hierarchy, an accelerator utilizes one of several *coherence modes* – ranging from bypassing the cache hierarchy entirely with DMA to participating in the system's coherence protocol when equipped with a private L2 cache – that can be selected at runtime based on the workload characteristics and dynamic status of the system [67, 68, 69]. ESP provides several design flows for new hardware accelerators: at the RTL level; with C, C++, or SystemC with a HLS tool; or directly from high-level machine learning models using the open source HLS4ML tool [70, 71]. When utilizing these flows, the socket of an accelerator tile provides platform services for address translation, DMA, configuration registers, and coherence. Hence, designers can focus on the optimization of

Figure 2.13: ESP Graphical Interface.

their accelerators without having to “reinvent the wheel” with respect to these capabilities. ESP also provides a ow to integrate pre-designed third-party accelerators (e.g., the NVIDIA Deep Learning Accelerator (NVDLA) [72]), so long as they comply with a standard interface, such as Advanced eXtensible Interface (AXI) [73].

The Network-on-Chip comprises six physical planes [61] and uses dimension-order look-ahead routing. Three planes are dedicated to coherence messages, two to DMA, and the last to access memory-mapped registers and interrupts. The NoC is synchronous and achieves single-cycle latency between adjacent routers. Each tile has a set of proxies that convert bus requests to NoC messages and vice versa. Thanks to the proxies, components in different tiles can exchange messages as if they were connected to the same bus.

ESP provides a graphical interface, shown in Figure 2.13, to guide designers through an interactive SoC design ow that allows them to: choose the number, types, and position of tiles, select the desired processor core among those available, determine cache hierarchy con guration, select clock domains, enable desired system monitors, con gure the size of NoC DMA plane. The graphical user interface (GUI) writes a con guration le that the ESP build ow can include to generate RTL sockets, the system memory mapping, NoC routing tables, the device tree for the target processor architecture, software header les, and con guration parameters for the proxy components.

The platform uses make targets to generate bitstream for the supported FPGA boards, compile Linux, create a default root le system that includes accelerators' drivers and test applications, and run full-system RTL simulations of a bare-metal program. The UART serial interface also enable the designer to run bare-metal applications on FPGA

Chapter 3: Scalable Methodology for Agile Chip Design

This chapter presents a flexible, robust and scalable methodology for the agile physical design of heterogeneous SoC architectures. As shown in Fig. 3.1, the methodology builds on the ESP platform for SoC design and programming [29]. The ESP platform combines a tile-based architecture and a system-level design methodology for the integration of OSH components and the derivation of the RTL implementation of a complete SoC instance. Thanks to the contributions described in the following pages, a GDSII implementation of each tile of this SoC instance can now be individually obtained from the RTL implementation by using the tile-based physical-design flow presented in Section 3.2. This flow relies on the enhancements to the ESP architecture described in Section 3.1. The top-level SoC integration, described in Section 3.3, uses the tile's ILM and LEF views for top-level placement and timing closure. Top-level DRC and LVS confirm the chip is ready for manufacturing. These steps can be performed in a more agile way with the verification and testing framework described in Sections 3.4 and 3.5, respectively. The proposed methodology was presented at the International Conference on Computer-Aided Design (ICCAD) in 2022 [39] and played a key role in the design and tape-out of two SoC architectures of growing complexity, as detailed in Chapter 4.

3.1 Enhancements to ESP for ASIC Design

For most of its history, the ESP project has focused on system-level design up to RTL implementations and SoC prototyping with FPGAs. Thanks also to the contributions described in this section, ESP can now be used to realize ASIC implementations of SoCs. The physical-design flow for agile chip development presented in the next sections is supported by many enhancements to the ESP architecture. This section summarizes the main ones.

Figure 3.1: Proposed methodology for agile chip development.

Hierarchy Restructuring. Previously, the NoC in ESP existed as its own entity – instantiated along with all of the tiles in the top level of the design. This design choice can pose problems for the top-level design and integration of a chip, as it can be difficult for EDA tools to achieve good QoR placing and routing of the NoC around the tiles. For this reason, we restructured the ESP RTL hierarchy, such that each tile now has a *wrapper* that instantiates the ESP socket, the encapsulated intellectual property (IP), and a single NoC router for each plane; this combination will henceforth be referred to as a *tile*. This approach better supports design partitioning (Section 3.2) and simplifies the top-level chip integration to routing connections between tiles (Section 3.3).

Control and Status Registers. Since our chip implementations require a greater degree of configuration at run time, we created a new set of control and status registers (CSRs) in each tile. Each set of CSRs is subordinate to an AMBA Advanced Peripheral Bus (APB) - an ARM bus protocol - that can be accessed either through ESPLink or from software running on the SoC. CSRs provide the configuration for components, such as a digitally controlled oscillator (DCO) and pads. Another important CSR is the tile ID register. The tile ID identifies the tile's position in

the SoC to logic within the tile. This is needed to form NoC packets and correctly route packets that pass through the tile. Previously, the tile ID was a parameter passed to the RTL of the tile at design time. However, in order to ensure that each type of tile only needs to be implemented once, we changed the tile ID to be in a CSR that gets set at runtime. At reset, a simple component in the auxiliary tile sends the tile ID over the NoC to each tile in a speci c sequence that guarantees the routability of all of these messages. The CSRs also include an extensive set of performance counters that monitor events such as NoC injections, coherence messages, cache hits and misses, external memory accesses, and more.

Local Clock Generation. Most ESP FPGA-based prototypes of SoCs utilize two primary clock sources: one for the NoC and memory tiles and another for processors, accelerators, and the auxiliary tile. Depending on the FPGA, the clocks can come from double data rate (DDR) controller IPs included in the design or from clocks provided by the FPGA's board. It is possible to instantiate phase-locked loops (PLLs) to provide an independent clock source to each tile [74]. However, depending on the number of tiles selected to use this feature, the available clocking resources of the FPGA can either be prohibitive or negatively impact the QoR of the system. Since ASIC implementations may desire independent voltage and frequency control for each tile and the NoC, we added the support for a DCO in each tile and a NoC DCO in the auxiliary tile. The DCO is designed for the target technology and is con gured with a dedicated CSR. Reset-generation logic is added to each tile with a DCO, so that reset is only deasserted after the DCO has been running for a certain number of cycles. This per-tile clocking scheme enables communication synchronous, globally asynchronous, locally synchronous (CS-GALS) approach, where all tiles run at an independent frequency and communicate through a synchronous NoC [40]. This strategy is key to the physical design ow, as detailed in Section 3.2.

Memory Integration. ESP offers a variety of on-chip memories for SoC designs [29]: 1) caches (L1, L2, and LLC); 2) shared local memories 3) accelerators' private local memories; 4) BootROM in the Auxiliary tile; and 5) dual-port register les in the Ethernet MAC. When targeting an ASIC implementation these memories, which are usually mapped to BRAMs in FPGA-based

Figure 3.2: Memory integration overview.

prototypes, had to be remapped to technology-speci c static random-access memories (SRAMs). A new memory integration ow was developed so ESP users don't need to master the platform database, les, and architecture hierarchy to remap the design into a different technology. The memory integration ow, shown in Fig. 3.2, consists of a memory list le, memory wrappers, a SystemVerilog le that maps the wrappers into the ESP architecture, and a directory location to link technology-based Verilog views. The memory list le (Fig. 3.2 (a)) speci es the ESP memory type (e.g., cache hierarchy level, IO tile bootram, accelerators private local memories (PLMs), and SLM tile memory), the technology memory's module name and area, and the wrapper's address size, word size, and port type. After this le is created, make target generates all wrappers compliant with ESP interface (Fig. 3.2 (b)), and the ESP mapping le for correct design instantiation (Fig. 3.2 (c)). The user needs to complete the technology-dependent memory instantiation inside each memory wrapper le. Standalone testbenches are automatically generated so users can verify the correctness of their wrappers before trying a full SoC RTL simulation.

For detailed information on memory integration, Appendix A presents a complete tutorial on

the technology-dependent components integration into ESP, including not only memories but also GPIOs and DCOs.

FPGA-link for DRAM Access. Due to the limited availability of open-source DDR controllers and the dif culty of including one in an academic tape-out, we chose to utilize an FPGA link as the primary source of external memory access. In each memory tile, the DDR controller that exists in FPGA-based ESP designs is replaced by a bridge from the SoC to an FPGA board, which hosts DDR controllers and the external dynamic random-access memory (DRAM). The FPGA bridge follows a simple credit-based protocol, with the FPGA host providing a clock to synchronize communication with the ESP chip. Messages are transmitted in the form of ESP NoC packets, allowing the reuse of components from the SoC in the design of the FPGA host.

SoCProbe for Single-Tile Test. In the context of chip manufacturing, ESP demands a robust pre- and post-silicon tile-based test strategy, decoupled from the system interconnect. The need for a NoC-independent Test Access Mechanism (TAM) for the SoC tiles is tightly related to the complex multi-plane NoC architecture of ESP: in case of logical bugs or physical design-based artifacts affecting the functionality of the NoC, an alternative way to communicate with the tiles is critical for diagnosing issues or independently testing the contents of the tile. Hence, we designed a SoCProbe Debug Unit to provide a new platform service, integrated in the ESP tile's socket, which enables a direct access to the tile leveraging four dedicated pins: SPI, SPO, SCLK, and SPM [75]. The SPM is asserted to connect the tile to the SoC Probe module in test mode and is deasserted to restore the standard connection to the NoC router in normal mode. The SoCProbe module is designed to bypass the NoC and, at the same time, mimic its latency-insensitive protocol [76, 62] while communicating to the tile. No additional modi cations to the tile's internal logic are required to support the test mode. Section 5.1 details the architecture and enhancements made to also support NoC testing.

3.2 Tile-Based Physical Design

As mentioned in Chapter 2, physical design flow has well defined steps (logic synthesis, floorplan, placement, routing, DRC, and LVS) that remain similar across the different offerings of EDA vendors. This section explains how our methodology provides productivity gains to a critical subset of these steps through flexibility, robustness, and scalability.

We start by adopting design partitioning, i.e., dividing the top-level design into smaller components. Design partitioning has several advantages in physical design compared to a full design implementation, including: the ability to parallelize the implementation by distributing the partitions to less powerful machines (scalability), a shorter time to execute the flow, and an easier way to detect and solve issues (robustness). The main drawback of design partitioning is its negative impact on the difficult task of achieving timing closure of the top-level synchronous interfaces. Since ESP tiles share the same tile-to-tile synchronous NoC router interface, the interface delays from one tile to another can be made very similar with a proper choice of the timing and floorplan constraints. This approach enables easier timing closure of a synchronous NoC and avoids the need to adopt an asynchronous NoC [77], which might incur a higher latency penalty.

Timing Constraints. Independent of the total number of tiles in ESP-based SoC, there are only four types of constraint files in the flow: AUX, MEM, SLM/CPU/Acc, and the Top-Level. As Fig. 3.3 shows, the AUX and MEM files must include external interface constraints that are design dependent because their external delay values must be set according to the environment (e.g. packaging and board) to which the chip will be exposed. The Top-Level constraint file shares the external delays set into the AUX and MEM files with a minor discount to account for the delay from the tile to the I/O pad. The remaining external interface constraints can be set according to protocol's standards (e.g. Ethernet, UART), with the exception of the clock frequency for the FPGA bridge in the Memory tile, which varies according to the test environment. Since the tiles use the same clock strategy and NoC routers, the setting of the clock relationships and all tile-to-tile external delays is design independent. There are only two design-dependent values common

Figure 3.3: Required methodology timing constraints.

to each constraint le type: the NoC clock frequency and the individual IP clock frequency. All technology-dependent constraints (e.g. driving cells, clock transition, clock buffers, etc.) must be set, but do not change across the top level and all tiles. In the constraint les, these constraints are speci ed with variables that are the mapped to the technology-speci c parameters in another le detailed in Appendix??. These settings are reusable across different designs that target the same technology process.

An architecture that allows for pre-de ned constraints signi cantly reduces the chances of mistakes in de ning timing constraints such as false paths, interface clock relationships, and incomplete constraints (thus enhancing the robustness of the design ow). Moreover, as the design scales, the only values that must be speci ed are the IPs' clock frequencies (highlighting scalability). The small number of constraints that must be changed for a new design or technology demonstrates the exibility of the ow.

Power Constraints. All tiles can support multi-supply voltage (MSV) and power shut-off (PSO) The use of a CS-GALS design naturally de nes the boundaries of the different power domains inside the tiles. The power domains follow the clock domains: the IP operates on the