Tampere University

ANTTI RAUTAKOURA

System-on-Chip Design

Can it be Agile?

Tampere University Dissertations 1395

Tampere University Dissertations 1395

ANTTI RAUTAKOURA

System-on-Chip Design
Can it be Agile?

ACADEMIC DISSERTATION
To be presented, with the permission of
the Faculty of Information Technology and Communication Sciences
of Tampere University,
for public discussion in the FA031
of the Festia, Korkeakoulunkatu 8, Tampere,
on 19th of December 2025, at 12 o'clock.

ACADEMIC DISSERTATION
Tampere University, the Faculty of Information Technology and Communication Sciences
Finland

Responsible supervisor and Custos
Professor
Timo Hämäläinen
Tampere University
Finland

Pre-examiners
Associate Professor
Luca Pezzarossa
Technical University of Denmark
Denmark

Associate Professor
Davide Rossi
University of Bologna
Italy

Opponent
Professor
Lauri Koskinen
University of Oulu
Finland

The originality of this thesis has been checked using the Turnitin Originality service.

Cover design: Roihu Inc.

ISBN 978-952-03-4303-3 (print)
ISBN 978-952-03-4304-0 (pdf)
ISSN 2489-9860 (print)
ISSN 2490-0028 (pdf)
http://urn.fi/URN:ISBN:978-952-03-4304-0

PunaMusta Oy – Yliopistopaino
Joensuu 2025

PREFACE

My path toward doctoral degree has not been the most typical. I earned my M.Sc. back in 2007. Immediately after that I was employed my Nokia Mobiles. Over the years, I participated in multiple System-on-Chip projects and I naturally learned a lot. As I gained more experience, I began to notice certain development habits that were not entirely optimal. I realized that I'm someone who is drawn to identifying problems and trying to improve suboptimal processes.

After spending 12 years in industry I felt ready to take next-step. Having previously given guest lectures in courses organized by Professor Timo Hämäläinen, I found it easy to reach out to him. I shared my observations on challenges in modern System-on-Chip development flows and proposed potential improvements. He was immediately interested, and I began my doctoral journey at Tampere University in 2019.

Around the same time, Professor Timo Hämäläinen launched major research consortium called SoC-Hub Tampere to boost regional competence in SoC development. This project was extremely important for my research, providing data to my thesis. I had the privilege of serving as project manager, which offered a unique opportunity to further expand my skill set. The development team, composed of talented recent graduates, was one of the most inspiring teams I've ever worked with—an experience I will cherish for the rest of my life. I would like to extend special thanks to D.Sc. Ari Kulmala, whose behind-the-scenes efforts were crucial in bringing SoC-Hub to life.

Having rather long career prior researcher does not mean that you are immediately ready to perform scientific work. That is something all researches needs to practice and Professor Timo Hämäläinen provided me very valuable lessons on scientific writing in the form of supervisor. Also I am very grateful form my colleagues who provided me very friendly working environment. You are too many to be listed but my special thanks go to Arto Oinonen, Matti Käyrä, Sakari Lahti, Joonas Multanen

and Esko Pekkarinen. I would also like to give my thanks to my pre-examiners Davide Rossi and Luca Pezzarossa for valuable and encouraging feedback.

My heartfelt thanks go also to my whole family -including my parents,sister, brother, spouse and two kids- who all have supported me throughout this journey.

ABSTRACT

Design methodologies and Electronic Design Automation (EDA) tools have improved over the decades to handle more complex designs, but the development process still resembles a conventional waterfall process where design activities are executed in sequence. Agile methodologies have become commonplace in the software (SW) development, but despite the success, agile methodologies have not been developed or applied to SoC development to a large extent.

This thesis addresses this gap from multiple perspectives. First, state-of-the-art was analyzed thoroughly with an extensive literature review and mapping study. The outcome of the study is that Agile SW Development methodologies have not been applied to System-on-Chip (SoC) development in a large extent and there is no clear agreement what agile development means for the SoC development. Instead, the research suggests multiple development methodologies to accelerate the development, and these contributions can be seen as a enablers toward Agile HW Development.

We applied and developed further the Agile HW Manifesto which originates from the Berkeleys's research. Based on our large SoC development project with three SoCs, we come in to conclusions that the most important management level principles is *Predictable schedule and fixed cadence* which is supported by implementation focused *Interface over instance* principle. The feasibility was proven by achieving one chip per year schedule for our SoCs.

To quantify the process, measurements were done on resourcing and the quality of the process was evaluated with the Fault-Slip-Through (FST) methodology. The results indicate that large SoCs in academic scale can be built within a year with a team of 20 developers and more precisely approximately 100 person months are needed. The use of FST showcases that it can also be used in HW context in addition to SW development and with our use-case, measurements indicate that 17 person months could have been saved if the bugs had been catch in the phase they belong to.

TIIVISTELMÄ

Suunnittelumenetelmät ja elektronisen suunnittelun automaatiotyökalut (EDA) ovat kehittyneet vuosikymmenten aikana käsittelemään monimutkaisempia järjestelmäpiiritoteutuksia, mutta kehitysprosessi muistuttaa edelleen perinteistä vesiputousmallia, jossa suunnittelutoiminnot suoritetaan peräkkäin. Ketterät menetelmät ovat yleistyneet ohjelmistokehityksessä, mutta menestyksestä huolimatta ketteriä menetelmiä ei ole kehitetty tai sovellettu järjestelmäpiirikehitykseen laajasti.

Tämä väitöskirja käsittelee tätä aukkoa useista näkökulmista. Ensinnäkin, nykytila analysoitiin perusteellisesti laajan kirjallisuuskatsauksen ja kartoitustutkimuksen avulla. Tutkimuksen tulos on, että ketteriä ohjelmistokehityksen menetelmiä ei ole sovellettu järjestelmäpiirikehitykseen laajasti, eikä ole selvää yksimielisyyttä siitä, mitä ketterä kehitys tarkoittaa järjestelmäpiirikehitykselle. Sen sijaan tutkimus ehdottaa useita kehitysmenetelmiä kehityksen nopeuttamiseksi, ja nämä ehdotukset voidaan nähdä mahdollistajina kohti ketterää laitteistokehitystä.

Sovelsimme ja kehitimme edelleen ketterän laitteistokehityksen manifestia (Agile HW Manifesto), joka on peräisin Berkeleyn tutkimuksesta. Suuren järjestelmäpiirikehitysprojektimme perusteella, jossa suunnittelimme kolme järjestelmäpiiriä, päädyimme siihen, että tärkein johtamisen periaate on *ennustettava aikataulu ja muuttumaton kehitysrytmi*, jota tukee toteutukseen keskittyvää *rajapinta yli instanssin* periaate. Menetelmän toteutuskelpoisuus todistettiin saavuttamalla yksi siru vuodessa aikataulu järjestelmäpiireillemme.

Prosessin kvantifioimiseksi tehtiin mittauksia resursoinnista ja prosessin laatua arvioitiin virheiden läpimenon (Fault-Slip-Through, FST) menetelmällä. Tulokset osoittavat, että suuret järjestelmäpiirit akateemisessa mittakaavassa voidaan rakentaa vuoden sisällä 20 kehittäjän tiimillä, ja tarkemmin sanottuna tarvitaan noin 100 henkilötyökuukautta. FST:n käyttö järjestelmäpiirikehitykseen osoittaa, että sitä voidaan käyttää myös laitteistokehityksessä ohjelmistokehityksen lisäksi, ja meidän käyttötapauksessamme mittaukset osoittavat, että 17 henkilätyökuukautta olisi voitu

säästää, jos virheet olisi havaittu kehitysvaiheessa, johon ne kuuluvat.

CONTENTS

List of Figures

List of Tables

ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
ART	Agile Release Train
ASIC	Applications-Specific Integrated Circuit
ASIP	Application-Specific Instruction-set Processor
BGA	Ball Grid Array
C2C	Chip-to-Chip
CD	Continuous Delivery
CGRA	Coarse-Grained Reconfigurable Array
CI	Continuous Integration
CPGA	Ceramic Pin Grid Array
CPS	Cyber-Physical Systems
CPU	Central Processing Units
DDR	Double Data Rate
DevOps	Development and Operations
DMA	Direct Memory Access
DRC	Design Rule Check
DSL	Domain-Specific Language
DUT	Design Under Test
EDA	Electronic Design Automation
eFPGA	embedded Field-Programmable Gate Array

FDD	FDD
FIR	Flexible Internal Representation
FOP	Feature-Oriented Programming
FPGA	Field Programmable Gate Array
FST	Fault-Slip-through
GUI	Graphical User Interface
HCL	Hardware Construction Language
HDL	Hardware Description Language
HLS	High-Level Synthesis
HW	Hardware
I2C	Inter-Integrated Circuit
IC	Integrated Circuit
IO	Input-Output
IP	Intellectual Property
ISA	Instruction Set Architecture
IT	Information Technology
KPI	Key Performance Indicator
LeSS	Large-scale SCRUM
LLM	Large Language Models
LP-DDR	Low-Power Double Data Rate
MDD	Model-Driven Development
ML	Machine Learning
MMIO	Memory-mapped Input-output
NPU	Neural Processing Unit
NVDLA	NVidia Deep-Learning Accelerator
ONNX	Open Neural Network Exchange
OOP	Object-oriented Programming

PCB	Printed Circuit Board
PD	Physical Design
PDL	Processor Description Language
PLL	Phase-Locked Loop
PPA	Power Performance Area
QFN	Quad-Flat No-leads
RISC	Reduced Instruction Set Computer
RTL	Register-Transfer Level
SaFe	Scaled Agile Framework
SDRAM	Synchronous Dynamic Random-Access Memory
SerDes	Serializer-Deserializer
SoC	system-on-chip
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SW	Software
TDD	Test-Driven Development
TX/RX	Transmitter/Receiver
UML	Unified Modeling Language
VLSI	Very Large Scale Integration
Xp	Extreme Programming

1 INTRODUCTION

Digital electronic devices have already shaped our lives for decades. Over time, these devices have become extremely powerful. For example, the Apple iPhone 12 operates 5000 times faster than the GRAY-2 supercomputer, the fastest computer in the mid-80s. The advancements in Integrated Circuit (IC) technology such as an increase in transistor density and architectural innovations have enabled the development of devices ranging from data centers and autonomous vehicles to smart wearables. In all these devices, the SoC remains the key component that provides the needed computing power and feasible power consumption. To be able to run different types of applications efficiently, SoCs typically include a mixture of programmable Central Processing Units (CPU), fixed functionality accelerators, and a wide variety of interface components. This heterogeneity has made SoCs extremely complex and developing takes a lot of resources and time. For example, NVIDIA announced their Xavier SoC for autonomous vehicles in 2018 and according to them, it took 8000 engineering years to develop [20]. Engineering effort associated with SoC design has become a limiting factor in the design of large SoCs [48]. At the same time, with the increased complexity, there is pressure to shorten the design cycles due to time-to-market pressure and cost efficiency of the development. The rise of open-source HW, especially the open RISC-V Instruction Set Architecture (ISA) based CPUs and open-source SoC platform components has also enabled academia to develop SoCs with a complexity not seen before in circumstances (limited budget, engineering resources, and time) similar to industry players. The aforementioned challenges require the design process used in development to be efficient and yield high-quality results.

In the waterfall process model, various design activities progress sequentially from requirements to implementation. This means that each step must be completed before moving on to the next step. Since requirements are established at the beginning, changes to these requirements are often viewed negatively, as they typically lead to

delays in the schedule. However, in the modern, fast-paced world, both technologies and market needs change rapidly, and requirement changes are likely. Due to the forementioned reasons, waterfall-based development is nowadays seen as unsuitable.

For almost three decades, SW development has overcome the situation by applying Agile SW Development methodologies. The essence of agile development methodologies is incrementality. The implementation is built peace by peace, and after each major increment, the implementation is demonstrated to the customer. Completion of each increment opens a natural point for requirement changes or new requirements. Because increments are rather short, it leads to better predictability of the schedule.

Although SoC development has similarities with software development (both have coding and testing phases), there are remarkably differences involved. SoC development includes tens of different tools, all requiring their own expertise, tool runtimes are long, which leads to fewer iterations, and most importantly, SoCs are a tangible product which must be fabricated. Fabrication is costly and takes time. Probably due to these differences and to our knowledge, agile SW frameworks such as SCRUM have not been applied to SoC development [68]. This may indicate that the differences between SoC and SW development are still so remarkably that known agile development methods cannot be applied directly to SoC development. All of this leads to a need to study the current state of Agile HW Development and to test agile approaches to a SoC design, which is the main contribution of this thesis.

1.1 Objectives and scope of research

The primary objective of this work is to propose an Agile HW Development methodology for SoC design and propose a way to measure SoC development process quality. The secondary objective of this research is to become a forerunner and key reference for SoC development process research. SW process research is very vigor with its conferences and journals, but similar research hardly exists in the HW domain. This research was done by executing a large SoC project with a roadmap of three chips, where the author contributed as a HW architect and project manager. The scope of this work is the HW development process. The SW development and the manufacturing process are left out of this work. Designs under study are big SoCs (in academic context) that implement a complete system instead of small discrete parts.

The research questions derived from the objectives are as follows.

- RQ1: What is the state of the art of the agile HW development?
- RQ2: What are the agile methods for SoC design?
- RQ3: How much resources are needed when SoCs are built with Agile HW Development methodologies?
- RQ4: How the development process quality can be measured?

1.2 Outline

This thesis is organized as follows. After the Introduction, research methodology is described in Chapter 2. A background is given in Chapter 3. The current state of the Agile HW Development is studied in Chapter 4 answering to RQ1. The SoC-Hub use-case is introduced in Chapter 5. Chapters 6 and 7 answers to rest of the research questions. Finally, conclusions are drawn in Chapter 8.

2 METHODOLOGY

2.1 Case study research methodology

Two main research methods are used in this thesis. The state of the art is studied with a rigorous literature review process as described in Chapter 4 for RQ1. Case study methodology [73] is applied for the rest of the research questions. The methodology is described in this Chapter. The use-case itself is described in Chapter 5.

Case study is a widely used research methodology in SW engineering. Runeson et al. define case study with the following definition:

Case study in SW engineering is an empirical inquiry that draws on multiple sources of evidence to investigate one instance (or a small number of instances) of a contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified. [73]

Although the definition is given in the context of software engineering, we do not see why the methodology could not be used in the field of computer engineering. Indeed, there are tons of publications on HW domain which use use-case methodology, but what is clearly missing is the accurate enough definition of the use-case strategy.

Common types of the use-case studies are:

- Exploratory: finding out what is happening, seeking new insights and generating ideas and hypotheses for new research.
- Descriptive: portraying a situation or phenomenon.
- Explanatory: seeking an explanation of a situation or a problem, mostly but not necessary in the form of a causal relationship.
- Improving: trying to improve a certain aspect of the studied phenomenon.

Our use-case study is exploratory. This type of study fits well to exploration of topics where the body of the literature is limited.

2.2 Research design

In this section, the research design is described in detail according to guidance provided in [73].

Rationale There is not enough knowledge about the applicability of the Agile HW development methods suitability in SoC development context. Research is also missing quantitative data about SoC design process so that true comparisons could be made.

Purpose We expect to increase knowledge about Agile HW methodologies and test its suitability to larger designs. This use-case study tries to strengthen or reject the hypothesis set by Agile HW Manifesto [53] by addressing research questions RQ2 and RQ3.

The case The case study is three SoC projects developed by the SoC-Hub consortium consisting of seven companies and Tampere university. The first SoC project was started from scratch and there was no existing platform to be used. Two following projects were executed in a row after the first. Altogether, the projects lasted three years, so the study can be regarded as a longitudinal study. The case study tries to find out if SoCs can be developed in an agile manner. In more detail, focus is in HW development and SW development is excluded.

Frame of reference There does not exist a dedicated frame of reference for the SoC development context. However, literary review can be used to form a foundation for case study [73]. A literature review regarding Agile HW development [68] has been published by the author.

Concepts and measures The entity of the case is the development phase of the SoC project. Quantitative measures include project duration, amount of human resources on different design domains, and process quality metrics. The gathered experiences are qualitative measures.

Methods of data collection and analysis The case study has been performed by the same person who acted as a project manager for the projects under study. Data for RQ2, was synthesized from these experiences. Data for RQ3 and RQ4 was collected from the version control repository and by interviews. Experiences were documented and analyzed post-hoc during the writing of the publications.

Case selection strategy The case was selected based on the availability and visibility of the data. Use-case studies are not common among the companies, and they tend to

protect their data.

Data selection strategy For this study, data selection is quite straightforward because there was clear connection between the data and research questions. Data for RQ2 and RQ3 were collected from all three projects. Data related to RQ4 was collected from the first project.

Replication strategy There is no previous studies to replicate.

Threads of validity There are multiple threads of validity. The main threads are: 1) Bias of the researchers. This thread is present because the development project and case study are run by the same researcher. 2) internal validity. The phenomenon under study could be affected by a third factor. As an example, project execution could have been fast due to some other factor than agile development. 3) External validity. SoC design is a popular topic in computer engineering, but the body of the literature for design process related is limited. This can lead to weak repeatability and a situation where drawing conclusions is challenging.

It is difficult to say based on a single study how much results can be generalized.

3 BACKGROUND

This chapter goes through the typical SoC development process and introduces both Agile SW and HW Development. Agile SW Development is covered, although SW development is out of the scope of this thesis.

3.1 SoC development process

SoC is composed of a HW structure and SW that is executed on the programmable units. An example of a recent SoC architecture is presented in Fig. 3.1 [70]. The SoC is composed of sub-systems, which are constructed from IP blocks. The design of these entities requires several design abstractions, starting from high-level models and ending up with RTL descriptions in Hardware Description Language (HDL) for Applications-Specific Integrated Circuit (ASIC) synthesis and layout placement. The design hierarchy is used to manage component reuse and enable concurrent development. SoCs typically form a programmable platform instead of being a strictly application-specific. For that reason, a SoC typically includes a wide variety of interfaces, ranging from slow-speed peripherals such as Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I2C) protocol to high-speed network connectivity.

SoC development process model is depicted in Fig. 3.2. The figure addresses design activities, timeline, and artifacts.

We recognize 13 different key **activities** involved in the SoC development. The activities are depicted in Fig. 3.2 as horizontal arrows. They include Modeling (1), Mixed-signal circuit design (2), RTL design (3), Synthesis (4), Physical design (5), Verification (6), Emulation and FPGA (Field Programmable Gate Array) Prototyping (7), Integrated Circuit (IC) Package and Printed Circuit Board (PCB) design (8), Manufacturing (9), Sample testing (10), HW Dependent SW (11), HW/SW integration and validation (12), Application SW development (13). Design tools are specific to the activity, which requires dedicated developer expertise.

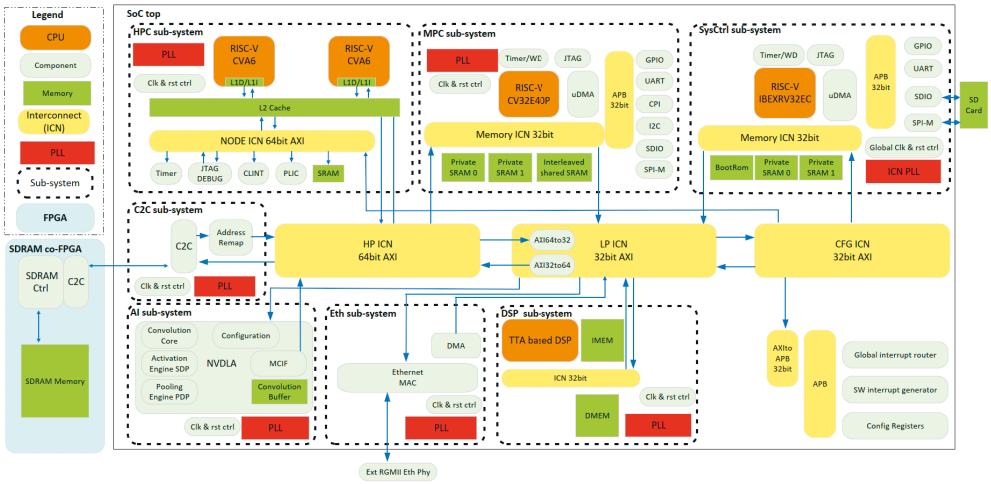

Figure 3.1 An example System-on-Chip *Ballast*. [70].

Milestones (HW0-HW9) consider the readiness of the SoC from the project start (HW0) to the validated physical chip samples with application SW (HW9). The milestones are used to communicate status across teams, synchronize the work between the activities, and act as quality control points. For project management, the milestones guide scheduling, resourcing, and decision-making in general.

The activities produce **Artifacts**. Artifacts evolve from high-level specifications toward more accurate implementations of the HW. Still, as of today, circuit diagrams are used for analog circuits and HDL code on RTL for digital design which are very low abstraction level models. Digital design is synthesized to gate netlist which is then placed and routed to layout and captured in GDSII format. FPGA synthesis produces bitstream to the FPGA platform. In SoC design, FPGAs are used for prototyping before fabrication of the SoC. The main artifacts for verification are test benches and test cases. SW is typically layered into a stack that consists of HW-dependent SW and application SW. After fabrication, the chip is first tested with sample tests to wake up the chip before execution of the SW stack. Sample tests are typically reused from verification and FPGA prototyping.

The SoC development process looks a waterfall at first glance, but development activities mostly overlap. Intermediate results of the work provide feedback, which leads to an iterative process. The concurrency and iterations also happen between design hierarchies (System, Sub-systems, IP components). Designs for large sub-systems are often run as separate sub-projects with their requirements, schedules, and

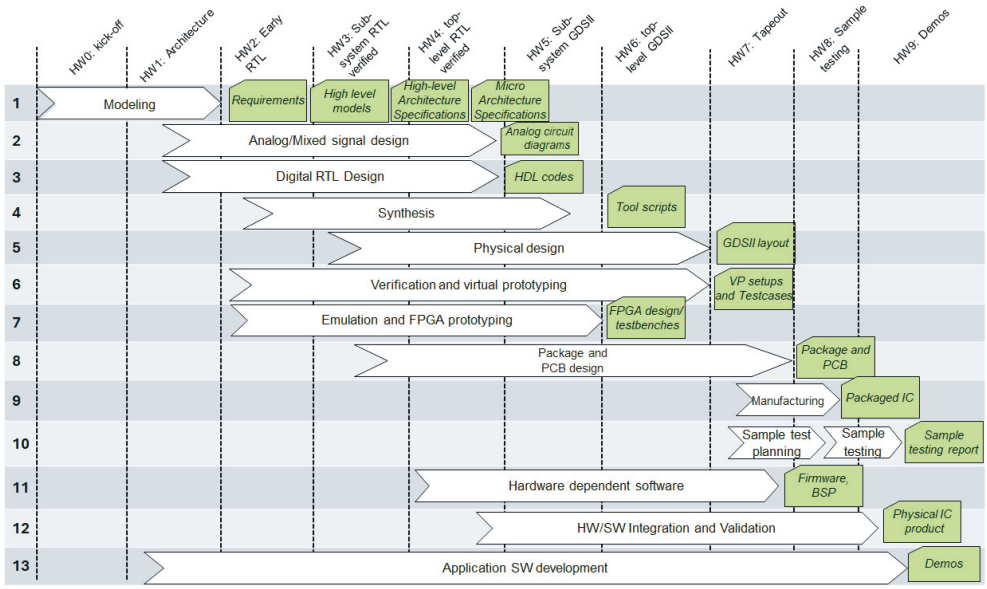

Figure 3.2 SoC development process: Artifacts, life-cycle, and activities.

resources. Despite the described parallelism of the work, HW design involves also challenging dependencies between the activities, which cause long feedback loops over the development cycle. For example, the netlist as a result of synthesis activity is dependent on HDL codes qualified by verification tasks. Design flaws are costly in human effort, calendar time, and money, especially when a new chip is needed to resolve issues. Implementation of the bug fix, re-running simulations, synthesis, and physical design can take weeks for a single design change and fabrication of the new versions takes several months.

Fig. 3.3 describes a typical SoC tool flow where the steps are grouped into four topics: Modeling, Design/Verification, Synthesis, and Physical Design (PD). The main steps have been established for decades, but naturally, tools have advanced to handle bigger and more complex designs.

Even though SoC flow steps have a tight dependency and are presented in a sequence, some tasks can run in parallel, e.g. emulator compilation and RTL synthesis. More importantly, development teams must go through multiple iterations during the project to reach a qualified RTL design and GDSII database for fabrication.

Tool run times increase even further when we proceed toward GDSII, and for a Ballast-sized chip they are approximately as follows, but naturally estimated run

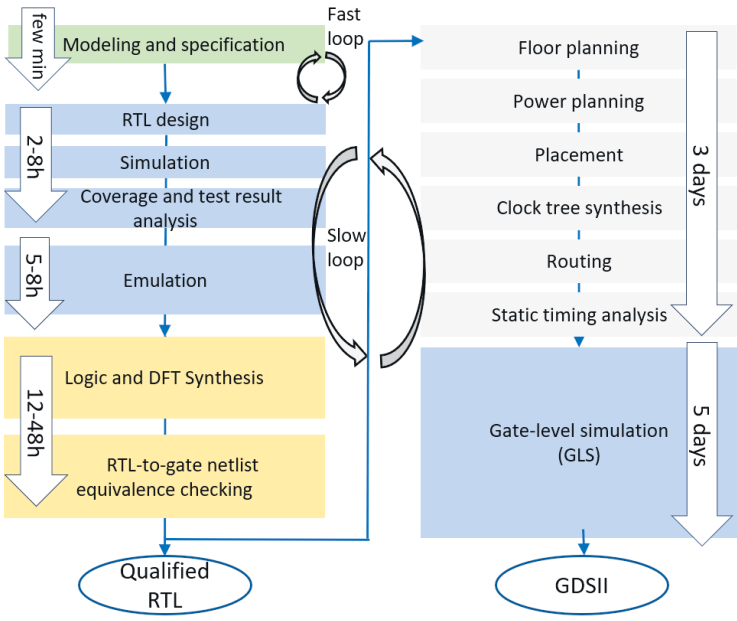

Figure 3.3 Typical SoC tool flow with tool runtimes.

times are design specific:

- RTL regression simulation 2 - 8 h
- Emulator/FPGA compilation 5 - 8 h
- Synthesis 12 h - 2 days
- PD 3 days
- Gate-level simulations (GLS) 5 days

The long feedback loop in SoC design is quite a game changer from typical agile SW practices, e.g. there is no way to make a verified RTL code release daily. The hierarchical design approach does reduce overall time, but still, synthesis and PD flow take 5–14 days, i.e. 1–2 weeks from small RTL change to GDSII delivery. Late changes become costly, and Qualified RTL is an important intermediate delivery and milestone.

3.2 Agile SW development

The *Agile SW Development* defines four well-defined values and twelve principles in the famous *Manifesto for Agile Development* [13]. That was a response and a criticism of traditional software development. The defined values are 1) *Individuals and interactions over processes and tools*; 2) *Working software over comprehensive documentation*; 3) *Customer collaboration over contract negotiation*; 4) *Responding to change over following a plan*. The given principles are the second layer of the manifesto, but still, those remain very abstract and do not provide practical guidance to implement them. Agile development is also often illustrated with a consecutive development loops visible in Fig. 3.4. Development consists of multiple similar iteration loops called sprints. During each sprint, an agreed set of features is implemented and released, and after that same is repeated until the customer is satisfied.

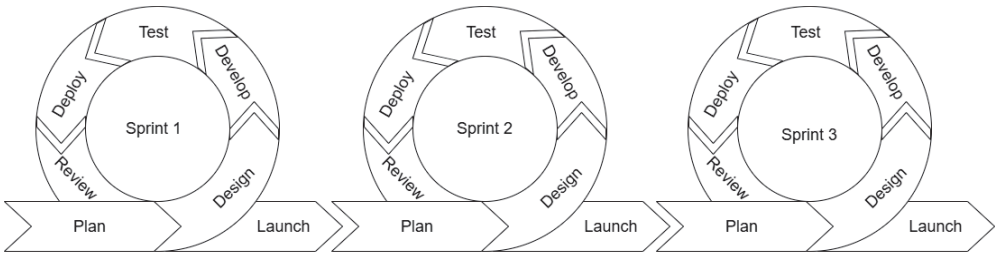

Figure 3.4 An agile development with three sprints. Adopted from [10].

Agile methodologies can be seen as a third layer of abstraction. They give more detailed guidance about how different values and principles could be achieved in practice. Scrum [77], Kanban [6], Lean SW development [66], Development and Operations (DevOps) [15] and Extreme Programming (xP) [12] are the well-known set of practices for running Agile SW development projects. Based on the large survey (2022) [26] Scrum and Lean were the most popular methodologies. Scrum focuses on project management practices [1], and Xp focuses also on the practical guidelines [12] for software development such as Test-Driven Development (TDD), incremental design, and Continuous Integration (CI).

The early work of Agile Software Development focused on small-scale software development, and the practices were addressing team-level work. As a consequence, large-scale agile has also been studied widely [27]. Scaled agile includes framework proposals such as Large-scale SCRUM (LeSS), Scaled Agile Framework (SAFe)

and SCRUM-of-SCRUMS and practices such as Agile Release Train (ART) [62]. Definitions of the project scale for large-scale projects measured with the amount of organization involved or engineering headcount varies, but in contrast, non-large-scale agile practices often refer to engineering headcount from 8 to 15 who are responsible for the complete SW project.

3.3 Agile HW development

Tailoring of agile development practices is common for different domains and types of projects. Kaisti et al. [43] (2013) notice the importance of process tailoring for embedded system development.

There has also been an effort to establish principles and practices for agile SoC development by UC Berkeley 2016 [53]. Their proposal includes four principles that are the baseline for this thesis. The values are following:

1: Incomplete, fabricable prototypes over fully featured models. Prototypes going through full RTL to GDSII toolflow allow accurate feedback on power, performance, and area. Also near tapeout ready design database potentially with a limited feature set provides the ability to adapt different tapeout deadlines, which accelerates system-level feedback. This principle is essentially about iterative Feature-Driven Development (FDD).

2: Collaborative, flexible teams over rigid silos. Engineers work in collaborative, flexible teams that handle all implementation stages. This is in contrast with the traditional setup with engineers assigned to particular functions. One key benefit is reduced communication overhead and decreased risks of misunderstandings.

3: Improvement of tools and generators over the improvement of the instance. Better design languages and tools are needed to improve the reusability and reconfigurability of the components and to reduce verification costs. Generators applied in the different design stages are needed to accelerate the development.

4: Response to change over following a plan. There is no doubt that requirements wouldn't change. Agile HW Development, however, sees changes to enhance competitiveness.

4 WHAT WE KNOW ABOUT AGILE HW DEVELOPMENT?

This chapter addresses the research question RQ1: What is the state of the art of the agile HW development? A mapping study and literature review are used as a methodology. The literature search was performed in two parts. The first search was done in 2021 and the results are published in [68]. The second search was performed in 2024. The presented results are based on the combination of these two datasets. This chapter includes sub-questions which address the main research question.

4.1 SoC development methodologies

This section gives an overview of the SoC development methodologies. We will focus on those that help categorize the mapping study and literature review results. This methodology overview is based on knowledge rooted in decades-long experience in SoC design in collaboration between Tampere University and industry.

The MDD is a top-down approach. The design starts from abstract models, and the flow continues through multiple intermediate modeling layers toward the target implementation. Each of the modeling layers adds more implementation-specific details to the model. MDD attempts well-defined model semantics, which enables correct-by-construct practices and formalism for code generation. The Unified Modeling Language (UML) is well known to model SW architectures before implementation. UML has also been extended to HW modeling with specific domain-specific metamodels and profiles. The IP-XACT standard is a modeling language for SoC development, and there are IP-XACT development tools such as Kactus2 [44]. In our previous work, we have demonstrated a multi-layer MDD framework for SoC HW development. [71]

The development platform, or development framework, is a collection of tools, repositories, and communication and documentation facilities for carrying out design

tasks in a team. We define the development platform so that it must include multiple design activities and not focus only on a single methodology. One example is RIVL platform with a highly automated design process, including RTL design, Integration, Verification, and Layout design [94].

The HW-SW co-design methodology focuses on design exploration of multiple implementation alternatives in the setup where HW and SW are developed concurrently [24]. The work with HW-SW co-design flows starts with an abstract model of the application, after which the goal is to find an optimal HW-SW division under given constraints. Executable system models written in programming languages are typically used because the exploration needs measurable results. The HW-SW co-design can be regarded as MDD practice, but the latter focuses mainly to design exploration.

The open-source HW is not a design methodology in the traditional meaning, but we have included it due to the significant boom after the releasing of the open RISC-V ISA and its open-source CPU core implementations. Open-source HW can mean open license-free standards, programming languages and libraries, design tools and tool flows, HW component libraries, or even open ASIC technology libraries. Open-source promises transparency, easier collaboration, and reduced costs, but from the development methodology point of view, the key question is the ease of reuse.

Platform architecture is a design methodology that allows substantial re-use of the software and can support a wide range of applications [47]. The system consists of HW platform which is "fixed" and includes basic building blocks of the SoC such as CPUs, DMA (Direct Memory Access), memory and connectivity peripherals. This HW is accessed through Application Programming Interface (API) which abstracts the HW for application development. Modern HW platforms are configurable and typically include domain-specific accelerators [5], [56].

LLM-assisted HW generation is a rather young technology to increase development efficiency. LLM-assisted SW generation is already a common practice [79] in the SW domain, and we expect the same trend to happen in the HW domain.

4.2 Related work

As far as we know, there are no prior mapping studies or literature reviews focusing on agile SoC HW development. Work done by Kaisti et al. [43] (2013) focuses on agile methods for embedded systems and embedded SW development. The primary result

is that the embedded systems domain includes problems that need to be solved before agile methods can be successfully applied. Reported challenges were a lack of tools, real-time constraints of the embedded systems, a need for HW-SW co-design, and a need for documentation to distribute information across different design disciplines. However, they also find common themes in successful practices which include TDD, CI, dual targeting (simulation before actual HW), iterative development, and customer collaboration.

The literature on embedded systems includes proposals on how to increase the agility of the embedded SW and the system development with methodologies such as HW-SW co-design [78] and platform-based design [19]. The proposed methodologies are also relevant for SoC development. Kaisti et al. also discuss embedded HW and ASICs, and they note that very little is known about Agile HW development as peer-reviewed academic work and there is a need for more rigorous research.

Demissie et.al (2018) [25] focuses on safety-critical embedded systems. The work shared eight references with [43] and included 14 papers published after it. The results show challenges such as HW development, team-based communication, and regulation process. Studied works applied selected agile practices, mainly from SCRUM and Xp. Some of these included studies addressed the need for tools support to effectively implement agile practices in the context of embedded systems.

The review by Ahmad [3] (2019) covers agile development of Cyber-Physical Systems (CPS). This work focuses on the presence of agile development in the CPS domain. The result is that related literature is limited, but the trend indicates an increasing interest.

To conclude the related work, SoC HW development has not been directly addressed in the prior literature reviews. Earlier works however discuss challenges and solutions relevant to the SoC HW development.

4.3 Research method

The research methodologies used in this study are systematic mapping study and systematic literature review defined by Keele et al. in the book *Guidelines for performing systematic literature reviews in SW engineering* [46]. The purpose of the mapping study is to show selected attributes among publications to provide an overview. For more detailed insight, the publications are analyzed thoroughly. The systematic process is

described in Fig. 4.1.

The study starts with the activity of formulating research questions (A1) in Fig. 4.1 and as an outcome, we came up with four different research questions (RQ1.1 - RQ1.4):

- RQ1.1: *What is the SoC HW development process coverage in the agile SoC HW development literature?*
- RQ1.2: *What agile methods or practices are currently applied to SoC HW development?*
- RQ1.3: *What methods and practices are proposed to increase the agility of the SoC development?*
- RQ1.4: *What is the impact of the work for the SoC HW development?*

RQ1.1 considers how much the literature covers the agile methods from the whole SoC HW design process. The key difference between RQ1.2 and RQ1.3 is that the former seeks the usage of known agile methods while RQ1.3 addresses the SoC development method proposals to increase agility. We noticed that it is important to make this separation due to the early age of agile HW development and the terminology that has not yet been established. The search strategy (A2) and filtering steps (A3-A5) are described in the following sections.

4.3.1 Search strategy

The purpose of the search strategy is to increase repeatability and quality of the work. The main task is to select search engines and databases and define the exact search string. The literature review was performed by using two search engines, Google Scholar and IEEE Explore. Google Scholar was used because it performs the search from multiple databases. IEEE Explore was used to make sure the other search engine results and get focused results.

Based on trial searches, we end up using the search string: *agile AND ("integrated circuits" OR ASIC OR FPGA OR SoC OR "system on chip" OR "system-on-chip")*. The term *agile* was selected instead of known agile methods in SW engineering, e.g. SCRUM. We noticed that including only the known agile methods led to the exclusion of many HW oriented papers. This is explained in more detail in the mapping study

results. The accuracy of our approach was verified by manually checking that a set of known publications were found in the search.

The search was done in two phases. The first search was done in 2021 and that covered publications from 2000 to 2021. This search was done for publication [68]. The second search was done in 2024 to complete the previous search.

Google Scholar yielded tens of thousands of search hits and IEEE explore gave a few hundreds hits. The remarkable difference is explained by the difference in the search algorithms. Google Scholar seems to be a heuristic. In practice, it means that after a certain point, the results do not any more represent the search string accurately enough. On the other hand, going through tens of thousands hits is not a feasible even at the title level. For that reason, an additional pre-filtering step (A3 in Fig. 4.1) was performed for the Google Scholar results. The results were checked in the relevancy order shorted by Google Scholar, just by reviewing the search results (title and context of the string match). The first 400 search results were checked at 2021 and 250 at 2024 respectively and included for further review. After that point, the relevancy shorting did not produce any more valuable publications for our purpose. At this point, the combined results from both databases included approximately 730 publications in 2021 and 450 publications in 2024.

4.3.2 Filtering

The filtering of the search results was performed in two stages (A4 and A5 in Fig. 4.1). The first stage filtering was based on title, abstract, and publication meta-data such as conference or journal name, while the second stage filtering was based on the full article review. Both filtering stages used the process of inclusion and exclusion [46].

The paper was included when all the following criteria were met:

1. I1: Paper addressed agile development of the SoC HW.
2. I2: Paper was published in a peer-reviewed academic forum.

The exclusion was done if any of the following criteria were met:

1. E1: SoC development was not addressed.
2. E2: The agile development was not addressed.
3. E3: The agile SoC development proposal was too specific to a certain type of

design or tool flow.

4. E4: Non-peer reviewed or non-academic publication.
5. E5: Duplicate of the same work.

The inclusion and exclusion set a clear scope for this work, increase robustness, and enable us to address the research question better. However, drawing the line on a decision for including publication to further analysis is not trivial. Examples of such borderline exclusions were a deep learning accelerator design flow specific for one type of neural network, and FPGA-based cloud computing accelerator specific for a dedicated big-data tool flow. Embedded systems and embedded SW-focused papers were the most common reason for exclusion. SoC HW was often mentioned in these papers, but the focus was not on the SoC HW development itself.

As discussed already, the term "agile" is a challenging search string. It was often referred to as a broad term, and we need to interpret the meaning. Typical exclusions were agile as a buzzword with a null meaning or an agile product feature instead of agility of the development process. For example, "a frequency-agile radio SoC" without any agility is mentioned in the design flow.

As an outcome, we had 150 (2021) and 45 (2024) papers after the pre-filtering and the first stage of filtering. Eventually, 25 papers were included in 2021 and 6 papers in 2024 after a full article review (A5). The exclusion criteria applied in 2024 are slightly different and that is discussed in the section 4.6. The selected papers are listed in Tables 4.2, 4.3 and 4.4.

4.3.3 Attributes

The research questions are addressed by the mapping study and the literature review. The data was organized by attributes defined in this section. The mapping study gives an overview of the results, while the literature review addresses research questions in more detail. The same publications and research questions are used for both methods.

The explanation of the attributes A1-A9 and abbreviations applied in the mapping study are given in Table 4.1. In all columns "NA" stands for "not addressed". The publications are indexed by attribute A1 in alphabetical order according to the first author.

The year of publication (A2) and publication forum provide insight into the

relevancy of the topic. For the mapping study, we recorded the engineering field (A3) of the forum to see which fields the agile HW development gets published in.

The SoC development process coverage of the publications is captured with attributes A4 (Management and life-cycle coverage) and A5. (Development activity coverage). These attributes address the research question RQ1.1. Due to the large scope of SoC development, it would be unlikely that a publication would address all aspects. Our interest is in observing where the current research focuses on and trying to find explanations for it.

The applied known agile methods and practices are recorded under the attribute A6. Known methods and practices originate from SW engineering, but we decided to do one exception here. The Agile HW Manifesto [53] is treated as a known methodology because it has been well recognized. This attribute addresses the research question RQ1.2.

Attribute A7 captures the methodology or practices to increase the agility of the SoC development. The purpose is to see what literature proposes to improve the situation. This attribute addresses the research question RQ1.3.

The impact of the work is evaluated by attributes (A8) and (A9). The scale of the project (A9) measures the development organization size. We defined the scale so that a small-scale project includes a single team of up to 10 members. large-scale projects consist of multiple teams or large teams beyond 10 members. The attribute A10 records the evaluation method, the institution type where the work has been performed, and the prototype implementation technology.

The reasons for selecting attributes A8 and A9 to judge the impact are: (1) The large project setup resembles a better SoC project setting with a complex design with multiple development activities and (2) Fabrication of physical chips guarantees that the design has gone through the complete SoC development process. Contributions with the hypothesis about the agile HW development without a use case are listed as proposals. Case studies performed by different types of institutions consider the development project with measured results. The research setup is captured from the reported affiliations. The key interest in the use case-type is to see the industry interest or if the methods are applied in the industry. The purpose is not to grade the impact between academic and industry work. Instead, the fabricated prototype gives

a clear measure of the impact. The fabricated chip is regarded as a high-impact work.

4.4 Mapping study

The mapping study data is illustrated in Fig. 4.2 - Fig. 4.7 where the attributes are on the X-axis and the frequency of the attribute is on the Y-axis. The mapping study results are presented by data recorded in Tables 4.2, 4.3, 4.4, and 4.5.

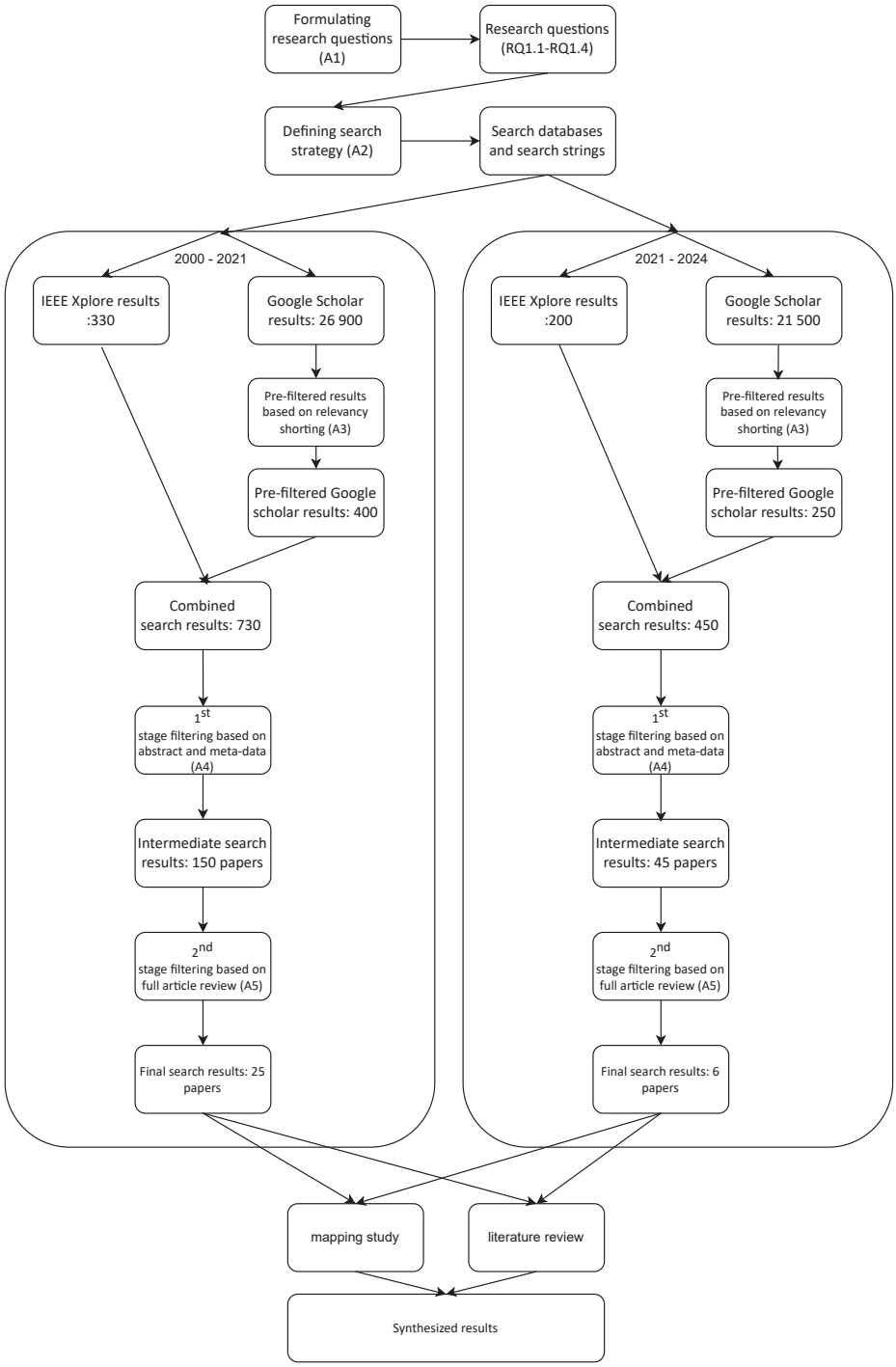

Figure 4.1 The mapping study and literature review process overview.

Table 4.1 Mapping study attributes and abbreviations. Attribute labels A1 - A10 matches to columns on Table 4.2 and Table 4.3.

Attribute	Description, abbreviations
A1: Index	The index (P1-P25) distinguishes literature review publications from other references.
A2: Year	The year of publication.
A3: Publication engineering field	The primary field of the publication forum: System (**Sys**), Hardware (**HW**), Software (**SW**)
A4: SoC project management and life-cycle level coverage	Which parts of the SoC process the publication address. Project management (**PM**), Exploration (modeling) (**L1**), Development (**L2**)
A5: SoC development activity coverage	Modeling (**1**), mixed signal design (**2**), RTL design (**3**) Synthesis and physical design (**4**), Verification (**5**), Prototyping (**6**), Hardware dependent SW (**7**).
A6: Applied agile methodology or practices	Recognized agile methods and practices. Also, proposals of the complement agile methods for SoC development are listed here.
A7: Proposed methodology or practices to increase agility	The attribute is synthesized from publications. Because terminology varies, the naming has been harmonized. The key practices include Code generation, Model transformations, Synthesis, Rising abstraction, Development platform, Platform architecture, HW-SW co-design, Open source or open access and AI.
A8: Scale	Small (**S**), Large (**L**). Determined by the size of the development team.
A9: Evaluation method, institution and prototype implementation technology	Proposal (**Prop**) i.e. not prototyped as a physical chip, Case study in: Academic (**CSA**), Industry (**CSI**), or Mixed (**CSM**) setting). Implemented prototype **FPGA** or **ASIC**.

Table 4.2 The list of included literature and the summary of mapping study results for publications P1-P7.

A1	A2	A3	A4	A5	A6	A7	A8	A9
P1, [4]	2009	HW	L2	3	NA	Rising abstraction, Code gen. for digital design	S	Prop.
P2, [5]	2020	HW	L1, L2	1-7	Agile HW Manifesto	Development platform, Open-source, Rising abstraction, Code gen.	NA	CSA, ASIC
P3, [9]	2019	HW	L1, L2	1-4	Agile HW Manifesto	Rising abstraction, Generator-based design, Code gen.	L	CSA, ASIC
P4, [8]	2020	HW	L1, L2	1, 3, 4, 7	NA	Rising abstraction, Code gen.	NA	CSA
P5, [11]	2020	HW	L1, L2	1, 3, 6	NA	Development platform, Rising abstraction, Code gen.	NA	CSI, ASIC
P6, [14]	2021	Sys.	L1, L2	3,7	NA	Development platform (ESP) [56]	L	CSM, ASIC
P7, [16]	2021	HW	L2	3-5	Agile HW Manifesto, 1) Complexity is your enemy 2) Do not fear refactoring 3) Do not over-engineer	NA	NA	CSI, FPGA

Table 4.3 The list of included literature and the summary of mapping study results for publications P8-P15.

A1	A2	A3	A4	A5	A6	A7	A8	A9
P8, [22]	2020	HW	L1, L2	1, 3-5	TDD	Development platform, Rising abstraction, Code gen.	NA	Prop.
P9, [28]	2021	HW	L1, L2	1, 3,5	Agile HW Manifesto	Rising abstraction (Chisel verification improvement)	NA	CSA
P10, [29]	2015	SW	L2	1, 6, 7	TDD	Rapid prototyping,Development platform	S	Prop., CSA
P11, [40]	2020	HW	L1, L2	1, 3, 4	Agile HW Manifesto, TDD	Development platform, Rising abstraction, Code gen., open-source tool interoperability	NA	Prop.
P12, [45]	2021	HW	L1, L2	1,3	Agile HW Manifesto	NA	NA	CSM, ASIC
P13, [50]	2016	Sys, SW	L2, PM	NA	SCRUM	NA	S and L	CSI
P14, [53]	2016	HW	L2, PM	1, 3-6	Agile HW Manifesto	Rising abstraction, Generator-based design, Open architecture, continuous and iterative digital design	S	CSA, ASIC
P15, [56]	2020	HW	L2	3, 6, 7	NA	Platform architecture, Open-source components, High-level Synthesis, templates and code gen.	NA	CSA, FPGA

Table 4.4 The list of included literature and the summary of mapping study results for publications P16-P21.

A1	A2	A3	A4	A5	A6	A7	A8	A9
P16, [59]	2020	HW	L1, L2	1, 3, 4	NA	Multi-layer synthesis flow from ML-models to HW description, rising abstraction	NA	CSA
P17, [64]	2020	HW	L1, L2	1, 3-5	Be tiny, be modular, be friendly (approachable)	Rising abstraction, Platform architecture, Open-source components, Evaluating design complexity and approachability	NA	CSA, ASIC
P18, [76]	2021	HW	L1, L2	1-5	Agile HW Manifesto	Code generation, Physical design tool abstraction (Hammer)	S.	CSA, ASIC
P19, [78]	2011	HW,	L1, L2	1, 3, 6, 7	NA	HW-SW co-design, Modeling, Model transformation, Code gen., HW emulation.	NA	CSA
P20, [81]	2020	HW	L1	1	NA	HW-SW co-design, Development platform, Modeling of heterogeneous system, LLVM compiler architecture.	NA	CSA
P21, [83]	2019	HW	L1, L2	1, 5, 6	NA	HW-SW co-design, Python-based development platform, Abstracting RTL Sim.	NA	CSA

Table 4.5 The list of included literature and the summary of mapping study results for publications P23-P25.

A1	A2	A3	A4	A5	A6	A7	A8	A9
P23, [86]	2014	SW	PM	1,3,7	SCRUM, Agile Release Train	HW-SW co-design, Sim models of the HW.	NA	Prop.
P24, [88]	2020	HW	L2	3-5	NA	Development platform for open source EDA tools, Incremental digital design, intermediate model for tool coupling.	NA	CSA
P25, [91]	2020	HW	L2	3-6	NA	Code gen., SystemVerilog coding guidelines, Wrapping of ASIC technology models	NA	CSI

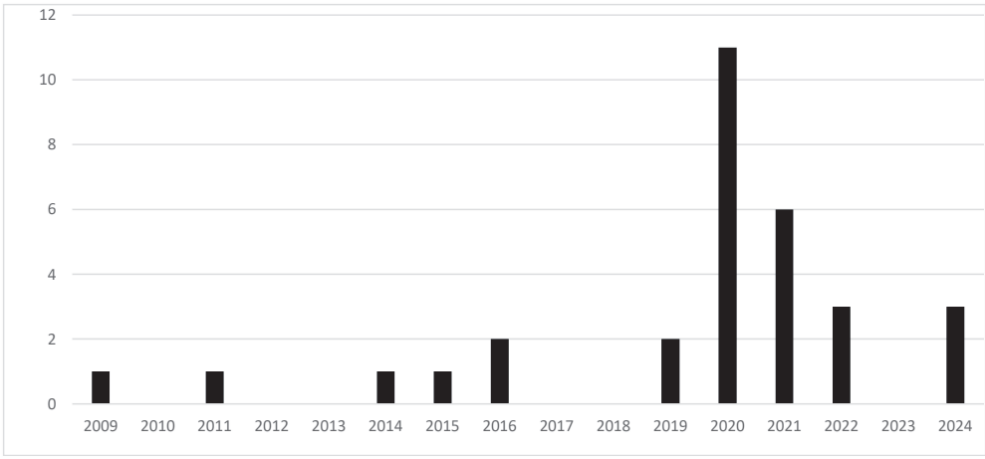

Figure 4.2 The number of publications per year (A2).

4.4.1 Overview

The timeline of the publications is presented in Fig. 4.2. The peak on 2020 is explained mainly by the IEEE Micro special issue on agile HW [41].

The publication forums and engineering field are listed in Table 4.6. We can notice highly ranked journals and conferences focusing on SoC designs being well presented. Most of the publications have been published in HW forums. It can be concluded that Agile HW Development has gained attention and the number of publications is large enough to draw conclusions.

4.4.2 RQ1.1: What is the SoC HW development process coverage in the agile SoC HW development literature?

The SoC process coverage data is presented in Fig. 4.3 and 4.4. A single publication can cover multiple process elements. From Fig. 4.3 we see that Exploration (L1) and Development (L2) life cycles dominate the results, and project management (PM) has low coverage. This is surprising because the most common agile methods (Scrum and Xp) in SW engineering are project management-oriented.

When we take a closer look at different SoC development activities in Fig. 4.4, we see that all activities have been covered at least to some extent. The modeling and the RTL design activities get most of the coverage. This might indicate that raising the design abstraction from traditional RTL models to high-level models is seen as

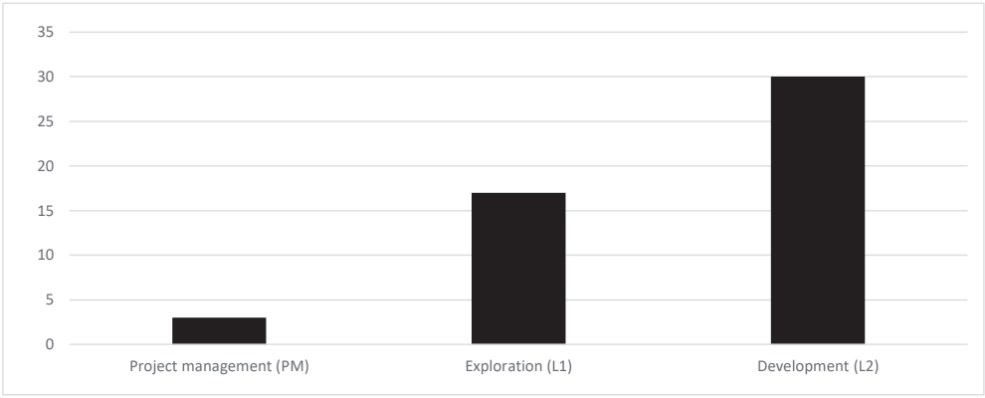

Figure 4.3 SoC process coverage (A4): Project management and life cycles L1 and L2.

attractive and feasible. The synthesis, physical design, and prototyping activities have moderate coverage, although these are very essential for SoC development in general. The results in verification are a bit surprising. Verification is commonly using programming languages or patterns, such as C or the object-oriented flavor of the SystemVerilog HDL. As a result, we would expect much more agile practices. The explanation could be the long feedback loops: RTL simulation is much slower compared to SW execution.

4.4.3 RQ1.2: What agile methods or practices are currently applied to SoC HW development?

Fig. 4.5 depicts applied agile methods and practices. The results show clearly that the traditional SW-rooted agile development methods and practices have not been applied to SoC HW development to a large extent. The reason for that is unknown and more research is needed to study this observation. The Agile HW Manifesto [53] seems to be currently the most famous methodology, but often that means usage of Chisel Hardware Construction Language (HCL) instead of following values defined by the Agile HW Manifesto. Two publications propose agile-minded guidelines. Bruant et al. [16] proposes, 1) Complexity is your enemy 2) Do not fear refactoring 3) Do not over-engineer and Petrisko et al. [64] proposes, 1) be tiny, 2) be modular, and 3) be friendly (approachable). These two are not counted as loosely defined agile methods. We also noticed that agile HW development has not been defined rigorously in most of the literature. Together, the results indicate that agile HW

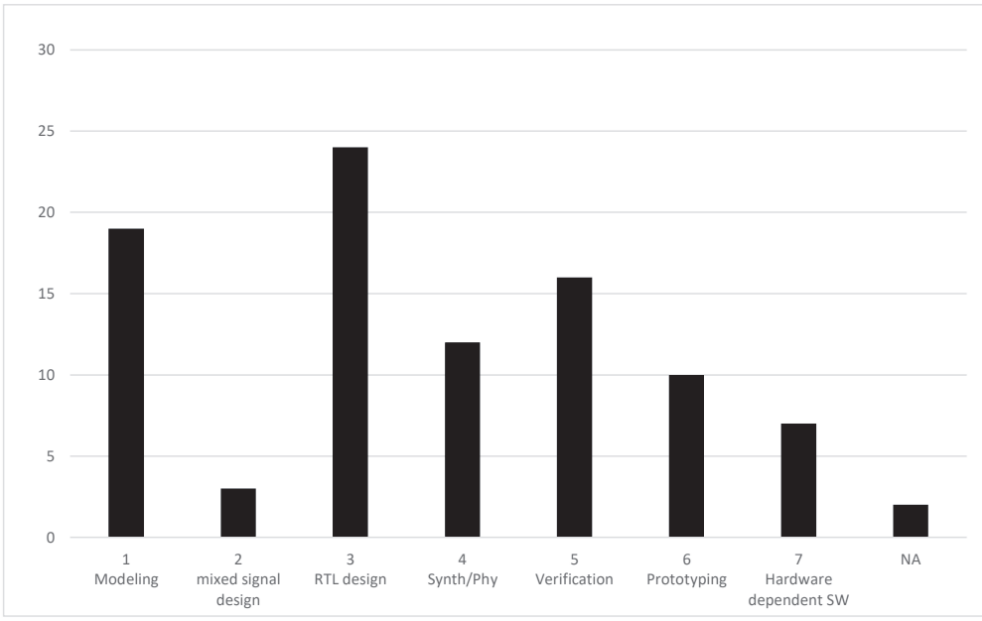

Figure 4.4 SoC development activity coverage (A5).

development is still in a very early phase. They did not either report problems or the unsuitability of the methodology.

4.4.4 RQ1.3: What methods and practices are proposed to increase the agility of the SoC development?

Fig. 4.6 lists various development practices and methods that have been proposed as a way to increase the agility of the development. The terminology and definitions vary between contributions, but conclusions are still possible from the collected data.

The most popular methodology or practice was related to MDD. Abstraction, code generation, model transformations, and synthesis were proposed as a way to increase the efficiency of SoC development. Rising the abstraction is essential to implement efficient code generation, model transformations, or synthesis. In a few cases, the increased abstraction was applied without automatic model refinements. It was seen as beneficial alone because such models give early feedback at the beginning of the development process on the known cost of accuracy. Other proposed practices included development platform, platform architecture, HW-SW co-design, open-source components, and LLMs. The proposals are analyzed in more detail as part of

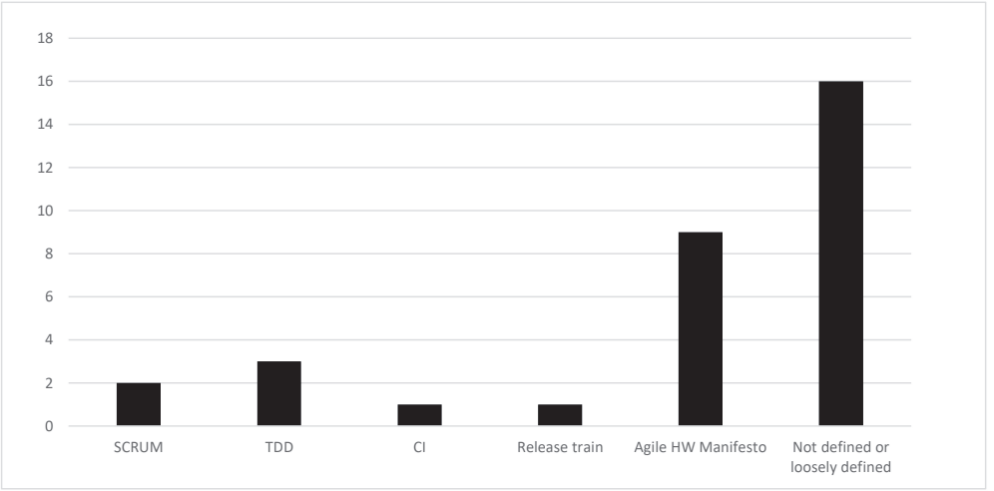

Figure 4.5 Applied agile development methodologies and practices (A6).

the literature review in section 4.5.

The important result here is that none of the proposals are novel in the SoC HW domain, and none of them are agile practices as such. The results indicate that the proposed methodologies and practices together with the agile mindset are seen as an enabler for agile HW development.

RQ1.4: What is the impact of the work for the SoC HW development?

The impact of the work was evaluated on three metrics: the scale of the work (Fig. 4.7), evaluation method (Fig. 4.8), and prototype technology (Fig. 4.9). The scale of the work was not typically mentioned, and the team was a small development team or a research group. Three were regarded as large-scale work. Case studies in academic setting were the most common research setup. The industry is contributing to research mainly through collaboration with academic partners. ASIC prototypes were reported in 13 papers. One reports 11 different chip prototypes [53], but our study regards it as one because the details of these prototypes are not shown.

In summary, the literature focuses on small-scale work in the academic setting, and the research favors practical work with prototypes over hypothetical proposals to follow the agile practices "*Working prototype over fully featured models*". It is very positive to see quite many ASIC prototypes that include the full design flow. It is well understood that those give the best impact, even ASIC prototypes are costly and labor-intensive.

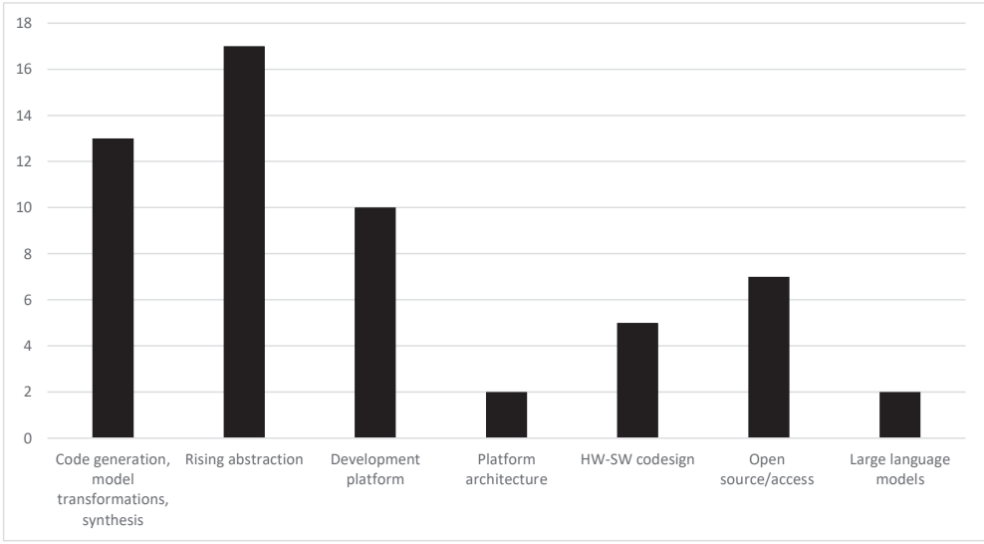

Figure 4.6 Proposed methodology or practices to increase agility (A7).

4.4.5 Summary

The mapping study gives an overview of the situation and addresses the defined research questions at a high level. The number of publications is gradually increasing. Contributions are mainly reported on HW research forums, and highly ranked journals and conferences were present to a significant extent.

The SoC design process was covered with healthy distributions and exploration and development life-cycles gets most of the coverage. The lack of a project management perspective is a clear gap. We also did an observation that the more activities focus on accurate models beyond RTL the less we get coverage in the literature. We believe that this is due to long runtimes and feedback loops in the tool flows that disfavor agile practices on quick iterations.

The Agile HW Manifesto, SCRUM and TDD were the most common known agile methodologies. However, in most cases the agile methods or practices are not defined, and popular methods on SW engineering such as SCRUM and Xp were not present to a significant extent. This can indicate that either HW developers are not familiar with the existing project management methods or those are not seen as feasible.

There are many proposed methods and practices to increase agility, but none of

Figure 4.7 Scale of the case study (A8).

them is completely novel or agile as such. Most of the works focus on increasing the development efficiency with MDD practices (abstraction, code generation, model transformations and synthesis).

The impact of the contributions is promising. ASIC prototypes are common and indicate that research is looking for practical solutions. Project organization, team sizes or other measures about the complexity were not typically presented and most of the contributions were done in an academic setting.

The research about agile HW development is clearly at early phase. Agile HW development was not defined clear enough in most of the publications. However, the results show challenges in the traditional development methods, and waterfall-driven processes. To improve the situation, rigorous research is needed to study the identified gaps.

4.5 The literature review

This section includes detailed analysis of the publications in the form of literature review. Especially, we will focus on the research questions RQ1.2 and RQ1.3. Because none of the selected publications provide quantified measurements of the design flow productivity, our contribution is to provide insights to the current state of the art on a detailed level.

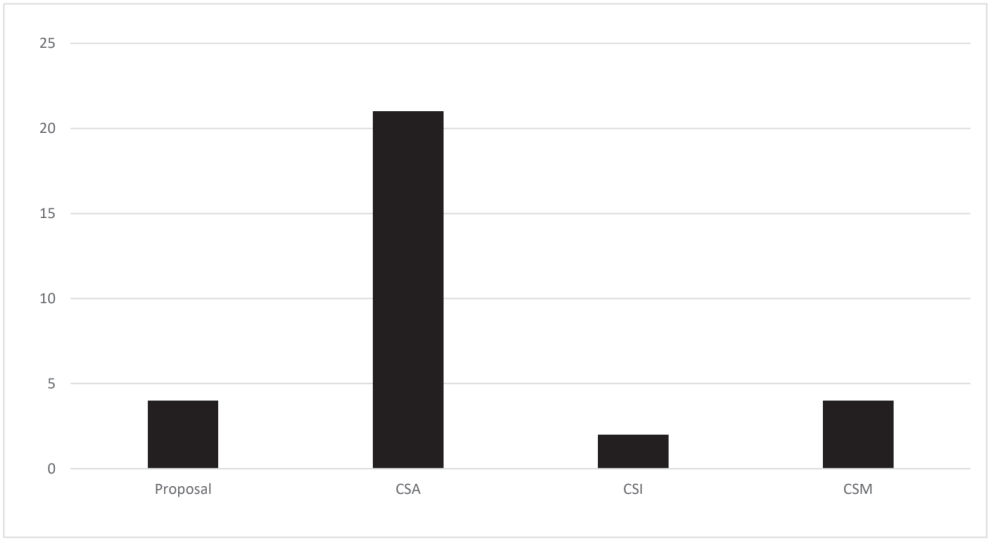

Figure 4.8 The evaluation method (A9).

4.5.1 RQ1.2: What agile methods or practices are currently applied to SoC HW development?

The SCRUM was the only applied methodology [86], [50]. The [86] is a tailored SCRUM proposal for Cyber-Physical Systems and covers SoC hardware development by proposing (but not implementing) simulation models of the HW for fast prototyping and HW-SW co-design. The key concepts discussed are separated HW and SW sprints, and integration of them with ATRs. As a result, the publication highlights domain-specific challenges such as HW-SW co-design, architecture, model-based development and verification, which could be addressed with the proposed agile methods.

Könnölä et.al [50] apply SCRUM to embedded system and radio frequency integrated circuit projects in the industry. Reported tailoring needs highlight challenges on the slow nature of the HW development and the different knowledge between developers in their disciplines. Different knowledge base makes the circulation of the tasks difficult, but team-wide planning sessions helped in understanding different areas. Applied agile methods did not increase productivity in the short term, but the increased team and system-level understanding could help productivity in the longer run.

Due to the nature of SCRUM, the [86], [50] naturally focus mainly on project

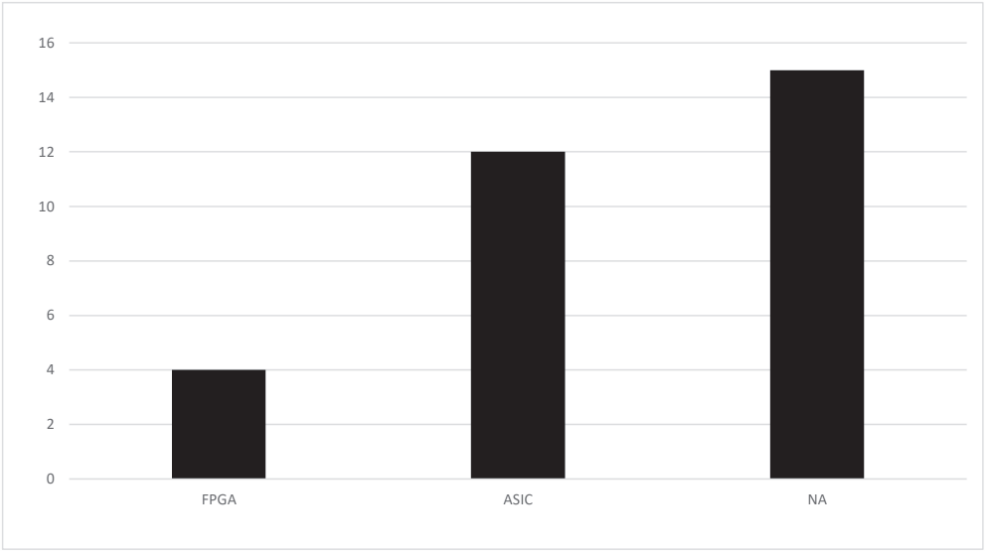

Figure 4.9 Prototype implementation technology (A9).

management and organizational topics. Highlights of the domain challenges are useful for applying agility to the SoC HW process.

The most common agile practice was TDD [29], [40] and [22]. To enable TDD for device driver development [29] showcases the usage of an abstract model of the HW by extending QEMU and KVM-based HW virtualization software. Memory-mapped Input-output (MMIO), DMA, and interrupts features were developed to increase the suitability of the development platform for TDD usage. The framework also works as an early HW model for rapid prototyping.

Jiang et.al [40] introduces PyMTL3, an open-source python framework for hardware modeling, generation, simulation, and verification. The PyMTL3 claims to enable the TDD by adopting the open-source pytest framework designed for unit testing of the python SW. The PyRTL [22] implements TDD by providing a toolkit where design and simulation (verification) happen in the same environment to avoid different tooling between these activities.

TDD was approached similarly in [29], [40] and [22]. The abstraction of the clock cycle accurate RTL model to a more inaccurate high-level model, or the level of the modeling language, e.g. Verilog to python was used to enable TDD. Abstracted high-level models are faster to simulate, and led to faster iterations, and the use of programming languages for HW modeling enables the use of popular SW development

Table 4.6 Publication forums and engineering discipline (A3).

Forum	Field	Amount
IEEE Micro	HW	8
IEEE/ACM International Conference On Computer Aided Design (ICCAD)	HW	3
IEEE Journal of Solid-State Circuits	HW	2
Design Automation Conference (DAC)	HW	2
IEEE Transactions on Circuits and Systems I	HW	1
IEEE/ACM International Symposium on Microarchitecture	HW	2
IEEE Computer	HW	1
IEEE International Conference on Electro/Information Technology	HW	1
IEEE Design & Test	HW	1
IEEE Nordic Circuits and Systems Conference (NORCAS)	HW	1
Conference on Programmable Logic (SPL)	HW	1
International Symposium of Electronics Design Automation	HW	1
IEEE/ACM International Symposium on Microarchitecture (MICRO)	HW	1
Design, Automation & Test in Europe Conference & Exhibition (DATE)	HW	1
International Conference on HardwareSoftware Codesign and System Synthesis	System	1
Workshop on Domain Specific System Architecture (DOSSA-3)	System	1
Journal of Systems and Software	System	1
ACM International Conference on Architectural Support for Programming Languages and Operating Systems	System	1
International Workshop on Rapid Continuous Software Engineering	SW	1
IEEE Software Testing, Verification and Validation Workshops (ICSTW)	SW	1
International Workshop on Rapid Continuous Software Engineering	SW	1

TDD tools for SoC development.

Lee et.al [53] promote free and open source RISC-V, Chisel Hardware Con-

struction Language (HCL), and hierarchical and automated physical design flow as practices to support the Agile HW Manifesto. An alternative implementation of the manifesto is in [40] in the form of Python-based modeling, generation, simulation, and verification.

The Agile HW Manifesto has gained much attraction despite its young age. Most importantly, multiple chip projects have been carried out. Lee et al. reports 11 chip projects, [9] reports large-scale (14 000 engineering hour project) usage, [45] reports a case study in a joint academic-industry setting, and [76] reports 8-core multiprocessor system with modern 16nm FinFET technology. [16] report usage in industrial organizations and also outlines additional guidelines borrowed from the agile SW development community: (1) Complexity is your enemy, (2) Do not fear refactoring, (3) Do not over-engineer. After 2021 Chisel penetrated the development of domain-specific components, but such work was excluded with exclusion criteria E3.

Lee et.al. [53] can be regarded as the key reference due to its proven impact and strong agile mindset with defined principles. On the other hand, our earlier research [70] indicates that similar efficiency and project execution time can be achieved by constructing agility to the top of the more conventional RTL to netlist methodology instead of modeling languages.

Petrisko et.al. [64] provides alternative principles for the HW community: Be tiny, be modular, and be friendly (approachable) based on their experience on how to increase agility. They also define four metrics to guide toward principal targets: quality, virality, efficiency, and functionality. *SW engineering of the HW* is also mentioned as the mindset to build agile HW Development. Although these principles are defined, they are rather abstract compared to the Agile HW Manifesto.

In a summary, the usage of known agile methods and practices in the HW domain is rare. The literature highlights challenges in adopting methodologies such as SCRUM, but on the other hand, TTD as a development-oriented practice can be seen as more feasible and beneficial. Early work of forming a definition for *Agile HW Development* is present to some extent, but still, the majority of the papers leave agile development undefined.

4.5.2 RQ1.3: What methods and practices are proposed to increase agility of the SoC development?

The following analysis is not an overview of the methodologies in general. Instead, it provides a view of the current state of agile HW development. One should be careful when interpreting the results. Although the review shows approaches to gain agility, applying these methods does not directly lead to agile development. For example, an SoC developed by HW-SW co-design methodology can still be applied using conventional, non-agile methods. As we noticed already in the mapping study, proposals often introduce the usage of the multiple methods. As a result, a single publication can be discussed in multiple subsections below.

The proposals are grouped into the following categories.

- MDD: Abstraction, model transformations, code generation and synthesis.
- Development platforms.
- HW-SW co-design.
- Open-source hardware.

Abstraction, model transformations, code generation and synthesis

Allen et al. [4] propose Rapid HDL based on the Microsoft .NET platform and utilizes .NET Common Language Runtime, which includes support for C#, Vb.Net, Ruby, F#, Cobal, and others.

The Chisel [53] is based on Scala language and thus inherits the modern SW language features such as parameterized types, abstract data types, operator overloading, and type inference. The Chisel generates an RTL HDL model of the HW, but the Chisel model can be translated to a cycle-accurate C++ executable for fast simulation. The Chisel's latest versions also include Flexible Internal Representation for RTL (FIRRTL) as a platform for writing circuit-level transformations. In addition to [53], the Chisel has been applied in [5], [9], [16], [28], [45], and [76], among the publications included to this literature review. Bailey et al. also utilize BAG2 [17], a process-portable circuit generator framework for analog and mixed-signal design, to decrease design time. [16] implements DDoS filtering HW. Although design is domain-specific, it has been included in this review because it was early work on testing Chisel's suitability to a different type of design. [28] and [94] extends the Chisel ecosystem by providing a verification toolkit. Karandikar et al. [45] implements Protocol Buffer [33] accelerator with Chisel. This work shows the industry's

(Google) interest in Chisel-based agile development. Schmidt et al. [76] showcase the applicability of the Hammer [87], a physical design generator, to decrease physical design time. Xu et al. [95] extends the Chisel ecosystem with verification and simulation tools.

Trapaglia et al. [83] presents DUTILS, a Python-based HW-SW co-design platform. The main tool is the DUTILS python class library, which abstracts the HW under simulation. It is used together with the CocoTB python class library for HW verification. The benefit is an incremental refinement of the Python-based system
models towards use-case-specific HW models. The proposal improves the reuse of verification environments and tests between the high-level models and RTL models.

The Python-based PyMTL framework is proposed by Jiang et al. [40]. PyMTL3 supports multilevel modeling on functional, cycle-accurate, and RTL. One of the key design principles is the modularity of the framework. That is achieved by dividing the platform into frontend, intermediate representations, and passes similar to the LLVM compiler architecture. The OpenPiton platform [11] also uses PyMTL based PyOCN network on chip generator in their work on developing custom SoC chips.

The PyRTL [22] is a python library for HW modeling similar to PyMTL3. The main difference is that it is a compact core library for HW development, which provides simplicity, usability, and clarity. The PyRTL can be extended with standard python modules, such as math modules to extend modeling capabilities. Verilog and Chisel FIRRTL code can be generated from PyRTL.

The Chipkit [91] uses a template-based code generation implemented in python to generate control and status registers for HW implementation with related SW API and documentation.

Mantovani et al. [56] use C/C++/SystemC class templates for different High-Level Synthesis (HLS) tools. Also, their design tool with a graphical user interface generates tile (sub-system) skeletons based on selected features.

Bahr et al. [8] present an HW-SW compiler framework to generate HW accelerators in Coarse-Grained Reconfigurable Array (CGRA) form. The primary input for the flow is a model written in Halide language. Halide is a C++ based Domain-Specific Language (DSL) for image processing applications.

Minutoli et al. [59] propose LLVM compiler based multi-layer synthesis framework to generate hardware descriptions of the machine learning accelerators. Multiple front-ends and back-ends can be supported. The framework is demonstrated with

Open Neural Network Exchange (ONNX) to Chisel FIRRTL synthesis through multiple optimizations.

Shalf et al. [78] combine model transformation and optimization for scientific applications written, for example in C++ or Fortran. They utilize commercial Tensilica Xtensa Processor Generator tool flow to explore different processor design choices to optimize applications and HW.

The DEC++ by Sorensen et al. [81] is a LLVM-based compiler for model transformations with related MosaicSim simulator to perform HW-SW co-design of the heterogeneous systems. Different target back-ends for multiple processor ISA such as x86 and RISC-V enables for architectural exploration of the system. The DEC++ does not produce synthesizable RTL HW models, but provides common SW API for custom HW accelerators.

Wu et al. [92] use "HW as a SW" approach to enable the usage of agile development practices such as CI and Continuous Delivery (CD) when designing Neural Processing Units (NPUs) for deep neural network interference. A Novel Chisel backend is used to create a clock-accurate model C++ of the HW. SW stack is based on Tensor flow. Together with the simulation model, it provides a fast purely SW-based HW-SW co-simulation environment which eventually enables usage of the CI and CD practices. This publication is a borderline paper. flow is specific for design of NPU, but this work is included because it contributes to the Agile HW Development with "HW as a SW" approach.

[99] demonstrates suitability of Feature-Oriented Programming (FOP) when designing microprocessors. FOB is an incremental design approach that is thus suitable to increase the agility of the development. Zou et al. extend PyRTL HDL with a design composition which is essential for FOP. The case study for evaluation is the development of RISC-V and OR1200 CPUs. Developers extended design with FOP, OOP, and direct modification methods by adding new instructions as a new feature. Development time with FOP is around half of the time used with Object-oriented Programming (OOP) and direct modification methods. Lines of Code (LOC) needed for implementation varies between the methods. LOC added by FOP are a few more than those by direct modification, but less than by OOP.

As a summary, high-level programming language-based abstraction, code generation and synthesis have been used to increase agility of the HW development. Chisel, python and LLVM compiler framework are common technologies among

the publications.

Development platforms

Amid et al. [5] introduces Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs. The Key tool is Chisel-based Rocket Chip RISC-V CPU SoC generator, FPGA-accelerated simulation with FireSim, and modular Very Large Scale Integration (VLSI) flow named Hammer. The Hammer provides an abstraction of the process technology and EDA tool-specific concerns, which enables easier change of the ASIC technology.

The OpenPiton [11] platform consists of reused Ariane Linux capable 64bit RISC-V CPU core, PyOCN network on chip generator compilation framework for open-source EDA tools, and FuseSoC IP management tool.

The ESP [56] is a Graphical User Interface (GUI)-based tool covering accelerator IPs and SoC design tool flows. The ESP platform address verification, FPGA synthesis, device drivers, and test applications based on design parameters defined by the user. On the SoC level, connectivity is also addressed in the form of generated routing tables, memory maps, device trees, and SW header files. The ESP platform has also been used for ASIC design by Bose et al. [14].

The Python-based development platforms [40], [22] and [83] gain mainly from the ability to develop modeling, digital design, and verification with the same input language and thus provide a unified development platform for developers.

Eschweiler et al. [29] base their platform on top of QEMU and KVM HW virtualization tools, which can address HW modeling and SW development.

The DEC++ [81] supports platform thinking through the use of the LLVM framework, which harmonizes different model transformations (LLVM passes).

Incremental synthesis and simulation framework LiveHD [88] is based on LLVM-like LGraph, a unified data model for open-source HW design. The work highlights the slow iteration time of the SoC simulation and synthesis tools as a key challenge for agility.

Xu et al. [95] applies the agile toolkit MINJIE to develop the modern high-performance superscalar out-of-order RISC-V CPU XIANGSHAN written with Chisel HCL. MINJIE includes a novel Diff-Rule based Agile Verification flow which allows CPU designs with different micro-architectural differences to be verified by the same reference model if they conform to the same specification, forming an

N-to-1 mapping between Design Under Test (DUT) and reference model. Another tool provided by MINJIE is Lightweight Simulation-SnapShot tool LightSSS for on-demand debugging. To speed up simulations simulation is run logging turned off. In the case of a detected bug, the tool returns to the previous automatically stored snapshot and turns logging on.

[94] introduces the RIVL platform to support the agile development process for processors. RIVL integrates a highly automated design flow for processor RTL design, Integration, Verification, and Layout design to improve processor development efficiency. RTL design flow is a mixture of reusable components in verilogsystemverilog and Chisel generated components. Verification of the CPU is based on instruction-level fine-grained online result comparison tool DiffTest and SoC design automation is achieved with Python scripts. Layout design is eased with configurable scripts.

Wei et.al. [90] proposes FPGA verification and debug toolkit to increase the agility of the FPGA-based validation by reducing the turn-a-round time of the FPGA flow. Toolkit includes efficient incremental compilation, SystemVerilog Assertion reuse from RTL simulation, and a SW-like debugger with features like breakpoints, watchpoints, and single stepping. The evaluation was done for 5400-core RISC-V SoC. Results show 18x speed up of compilation time when incremental compilation is in use. Synthesized assertions took a negligible amount of FPGA resources.

In a summary, we notice that development platforms are built around modeling with programming languages, integration tools to increase automation, or tool flows. In addition, extending some existing frameworks like the LLVM compiler is present. The code generation makes sense the best when multiple different targets are generated. Thus, the code generation and development platforms often exist in combination. Some platform architectures are present in [56] (ESP), and [64] (Blackparrot), but the border between the development platform and the platform architecture is not always clear. These "platform development platforms" come with a reference architecture as a starting point.

HW-SW co-design

HW-SW co-design based approach by Sorensen et al. [81] is based on a compiler that automates architectural exploration of heterogeneous systems. The flow supports popular programming languages such as C++ and python. Trapaglia et al. [83] bases HW-SW co-design support to reuse between the HW design and the

verification. Automation is not involved when refining from SW-based models to HW implementations. Instead, they talk about fluent migration from the models to the implementation. The strategy of proposals by Shalf et al. [78] is highly HW-SW co-design focused.

Wagner et al. [86] highlight the importance of HW-SW co-design for agile development, but do not present a detailed methodology to implement it.

Despite a relatively small number of publications, the typical HW-SW co-design flows are present. In a summary, the proposals focus more on exploration and high-level models rather than the development of lower-level implementations. The HLS approach proposed by [56] could be seen as HW-SW co-design, but the authors did not consider it in this category.

Open-source hardware

The open-source HW is present to a great extent, but we focused only on contributions that increase development efficiency and agility.

Lee et al. [53] outline free, open, and extensible RISC-V ISA as one of the keys for implementing their Agile HW Manifesto. Openness together with ISA extensions provides a possibility to explore different architectural changes with small teams. Also, the Chisel language is an open-source implementation.

The Hammer for digital design and the BAG for analog and mixed-signal are ASIC technology and tool-agnostic abstraction layers for physical design [5]. These tools help in exchanging flow-related information in an open-source manner. Exchanging design information in a commercial physical design tool format is often forbidden.

As the name suggests, OpenPiton [11] takes openness as its key discipline. They use and contribute open-source HW, and also share important insights to build and maintain such a community.

PyMTL3 [40] is released as open-source, but more importantly, from an agility point of view, emphasizes interoperability with other open-source hardware tools.

The ESP platform [56] contributes to the open-source HW community by providing an integration platform for heterogeneous systems to gain agility on the SoC level.

The Blackparrot [64] base their fast development iterations on open-source RTL component libraries BaseJump STL and HardFloat.

In a summary, openness is addressed from multiple perspectives. Often, the link

between agile development and openness is implied but remains weak. Open-source could potentially increase flexibility eventually by allowing faster and cheaper experiments, ease reuse and open entirely new possibilities by making SoC design affordable for a wider audience. However, we have identified in our previous work also quality issues in open-source HW such as lack of documentation and varying quality [70].

Large Language Models

Lie et al. [54] use ChatGpt LLM in Chisel and Verilog code generation to increase the agility of the HW design. As a demonstration, they implemented Chisel and Verilog versions of RV32I RISC-V CPU with 5-stage pipeline and dynamic branch prediction with the help of LLM. Design is validated with functional verification and physical design with Skywater 130nm technology node. As a result, Chisel version outperforms Verilog in terms of design correctness, area and power.

ChatCPU [89] is a LLM-based design and verification platform. ChatCPU flow includes the usage of novel PDL (Processor Description Language) to model CPU in machine-readable format. The model is used as a baseline design and reference model generation. The capabilities of the ChatCPU is demonstrated by generating a 6-stage RISC-V CPU which was fabricated with 130nm process node. Work also reports design average productivity gain of 3.81X when compared to manual design.

As a whole, when we return to *RQ1.3* we notice that literature addresses the challenges of SoC development time, and provides some solutions to improve it. However, the link to agile development remains weak.

In the summary, the usage of the methods has penetrated to new areas, but that causes solutions to be domain-specific and does not solve agile development of the SoCs. Also, a combination of different methodologies likely lead to too complex a development flow.

4.6 Notes

Selected publications have been brought into discussion, although being excluded from the literature review. Especially in the literature review update in 2024, we noticed a phenomenon that a number of new proposals started to saturate, and we applied exclusion criteria E3 (The agile SoC development proposal was too specific to a certain type of design or tool flow) more strictly. A couple of borderline papers

in this category were included in 2021, but that was due to the early days of agile HW development.

Too a narrow scope such as the implementation of CNN [36], [96] or showcases of the usage of the Chisel HCL to domain-specific designs [35], [57], [93], and [65] were excluded. [72] used Chipyard [5], but does not extend methodology with any means. The exclusion criteria E5 (duplicate of the same work) was applied to [39], [74] which overlaps with ESP Platform [56] and [51], [18], [31], overlaps too much with [8] and focuses on CGRAs only. In both of the cases, papers also originate from the same research group.

The question is that does Agile HW Development need these proposed methodologies to become agile? Now the claim is that by decreasing iteration time the project becomes more agile. We argue that there is no as strong link between as claimed. Agile methods on the SW side does not place programming languages into center, either. Still, usage of efficient development methodologies can lead to inflexibility in project management or development process. This thesis's main research question RQ2: What are the agile methods for SoC design? will delve into Agile HW Development in a broader context than development methodologies.

4.7 Takeouts

The number of publications and presence in high-impact forums shows increasing interest on Agile SoC HW development. The research seems to be in the early phase and not even agreed definitions exist compared to Agile SW Development. One explanation might be that HW designers are more artifact oriented and have not yet seen the importance of the development process. On the other hand, the long feedback loops and highly dependent activities narrow the space for agility.

We summarize our findings from the mapping study and literature review in the following by each research question.

- RQ1.1: The papers covered exploration and development life-cycles of the process, but hardly anything on process management.
- RQ1.2: None of the famous SW agile methods were literally applied to the SoC HW development. The Agile HW Manifesto is the most established set of principles so far.

- RQ1.3: MDD was not explicitly mentioned, but many papers addressed abstract models and code generation to improve productivity and quality. However, quantified results on the development efficiency are mostly missing. The latest methodology is LLM-based automation, but the potentially small amount of available training data compared to SW code might hinder the quality of this option.
- RQ1.4: Several ASIC implementations were found, which increases the impact of the work. Also, the idea of frequent prototyping fits well in agile development.

The literature demonstrates a strong presence of open standards, open-source component libraries, and development tools. Open-source HW will likely be the game changer. Open-source EDA tools already provide cheaper experimenting, with promising early results [42]. Commercial tools will stay de-facto for production quality designs, but open source helps the development community to freely exchange ideas and experiences. Agility can break out from the new developer communities whose members might not have HW design background.

5 THE USE-CASE: SOC-HUB SOCS

The use-case in this thesis is a multi-year SoC development project with a "one chip per year" target. The project organization is called SoC-Hub [84], and it is a university-led consortium with seven industry partners. One important aspect of the project was also to train new SoC talents. All three SoC projects went through the process from idea to functional system with supporting electronics in addition to chip design. The SoC-Hub chip roadmap and included main IPs are presented in Fig. 5.1. The message of the figure is that we started with a rather big design, new IPs were developed with the second SoC and the third SoC integrated all this together as a complete platform. Most of the work started after completion of the previous project, i.e. the development work for the SoCs was not pipelined. The project infrastructure was built from scratch in 2020 and the first chip project called Ballast was started in fall 2020. Project infrastructure included literally "everything" including repositories, tools, servers, access to 22nm ASIC technology library, and tens of agreements. In this chapter, all three SoCs are described in more detail.

5.1 Project setup

The SoC-Hub consortium formed a collaborative organization where all institutions participated in areas with the most expertise. Company members participated as supervisors or developers. The development team was a mixture of junior, senior, and expert-level developers. *Junior* means master's graduate or trainee, *senior* has at least three years and *expert* more than ten years experience. The average competence level was at the senior level and the project team size was around 20 persons. The exchange of company-developed IPs was handled in GDSII format or encrypted RTL.

The development team had a flat organizational hierarchy and SoC projects had the same project manager with a SoC development background. The project manager also acted as an architect. Physical design and SW had dedicated leads who also

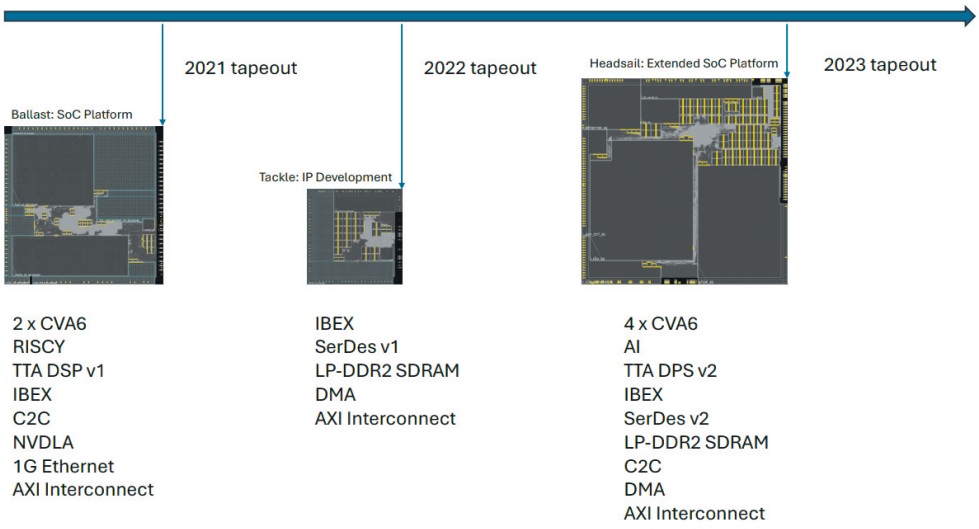

Figure 5.1 SoC-Hub chip roadmap.

participated in the development. FPGA prototyping had a dedicated team, but the rest of the team was organized based on the sub-systems instead of the design activity. The same developer participated in multiple design activities, RTL design, and physical design for example, which means that persons moved along with the design process. There were three weekly project meetings. An overall project meeting which focused on RTL design and verification in addition to general topics, a meeting for physical design as well as a SW meeting. Sub-system teams were so small (2-3 persons) that they handled their communication directly without meetings. One exception to this was the Deep Learning Accelerator (DLA) development team, with four persons and dedicated meetings.

Appropriate server capacity was set up by the recommendation of companies. SoC-Hub had 3 physical servers each having 2x Xeon-G 6248R 24-core CPUs running at 4GHz and each having 1,5TB of RAM. The disk server had 16 TB of disk space. Several virtual servers were used for simulation, code repositories, CI pipelines, physical design, and license servers. Documentation was kept very close to code by files in the Markdown format, and the documentation was rendered automatically to web pages. With these powerful machines, physical design runs from RTL to Design Rule Check (DRC) checked GDSII took 1 to 64 hours depending on the sub-system size and the longest GLS took even seven days to complete. Commercial EDA tools were used in all design activities.

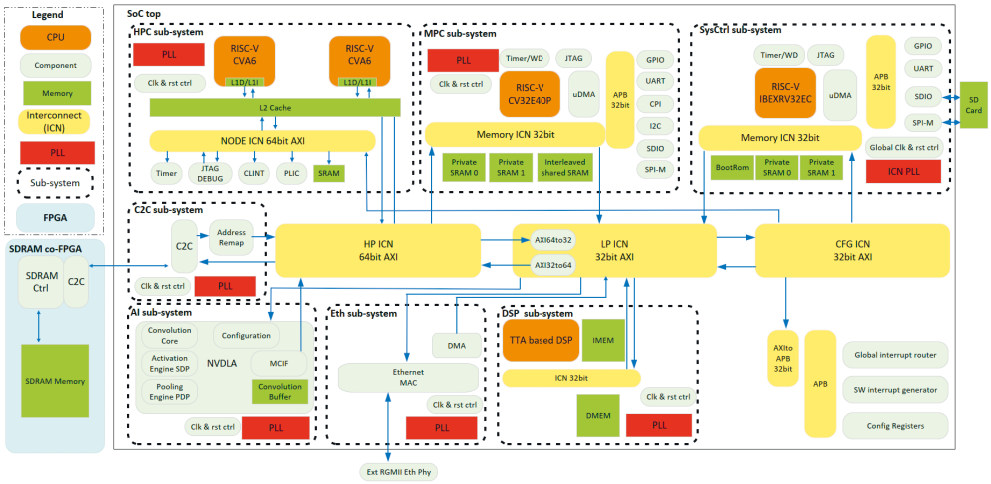

Figure 5.2 High-level block diagram of the Ballast SoC [70].

5.2 Chip features

The content of these three chips is only described on a high level so that use-case setup and results of this thesis can be understood. Platform-based design and reuse were heavily utilized in the SoC architecture. Reuse was possible due to the rise of open-source hardware projects, and we reused three different variants of the open-source Reduced Instruction Set Computer -V (RISC-V) CPUs originating from Pulp-project [100]. In addition to reused components, projects included multiple new IPs developed as a part of the SoC-Hub project.

The first chip is called Ballast, and it consists of seven different sub-systems [70]. The high-level block diagram is presented in the Fig. 5.2. The Ballast is 15mm^2 chip and holds 130M transistors. The die is in a Ceramic Pin Grid Array (CPGA)-255 IC package. The main purpose of the Ballast was to establish development infrastructure, test the quality of the open-source components, and design a functional platform that can be used as a baseline for forthcoming chips.

The Origins of different IPs in the Ballast project are following:

- Phase-Locked Loop (PLL) clock generator mixed-signal design. Designed by one of the collaborating companies (new IP).
- HPC sub-system: 2 x CVA6 RISC-V CPU [98] (reused), L2 Cache (new IP). fmax 500MHz, area 5,3mm^2.

- MPC sub-system: Pulpissimo [75] with CV32E40P RISC-V CPU (reused). fmax 500MHz, area 2,5mm^2.
- SysCtrl sub-system: Modified Pulp-platform with the IBEX RISC-V CPU [60] (reused). fmax 30MHz, area 0,35mm^2.
- AI sub-system: NVidia Deep-Learning Accelerator (NVDLA) [61] (reused). fmax 750MHz, area 2,6mm^2.
- Ethernet sub-system: Modified Librecores [32] 1G controller. (partly reused IP). fmax 125MHz, area 0,7mm^2.
- DSP sub-system: Application-Specific Instruction-set Processor (ASIP) generated with the Open ASIP TCE toolset [38] which is designed under SoC-Hub (new IP). fmax 600MHz, area 0,95mm^2.
- Chip-to-Chip (C2C) sub-system: Off-chip Interface IP to connect companion FPGA for Double Data Rate (DDR) synchronous Dynamic Random-Access Memory (SDRAM) access. (New IP). fmax 400MHz, IOs 100MHz, area 1mm^2.
- Interconnect sub-system: three-stage crossbar AXI interconnect. Build based on the pulp AXI library components. fmax 1,2GHz, area 2,37mm^2.

The second chip on the roadmap was the Tackle. Tackle is a 4mm^2 IP prototype chip with 12M transistors for essential platform IPs. Low-Power Double Data Rate (LPDDR)1, LP-DDR2, DMA, and High-Speed Serializer-Deserializer (SerDes) IPs were developed. Different LP-DDR versions were placed on two different variants of the chip. LP-DDR2 and SerDes IPs included analog macros which typically need multiple versions to reach targeted quality. Because Tackle included only a moderate amount of Input-Output (IO) pins, a standard Quad-Flat No-leads (QFN)-88 IC package was selected. While being an IP prototype chip it did not need much of computation performance, and SysCtrl was reused from the Ballast to perform boot and control of the designed IPs for prototyping needs. The architecture block diagram is presented in the Fig 5.3.

In summary, the Tackle IPs are following:

- PLL: Reused from the Ballast as a hard macro.
- SysCtrl sub-system: Same as in Ballast (reused). fmax 30MHz, area 0,35mm^2.

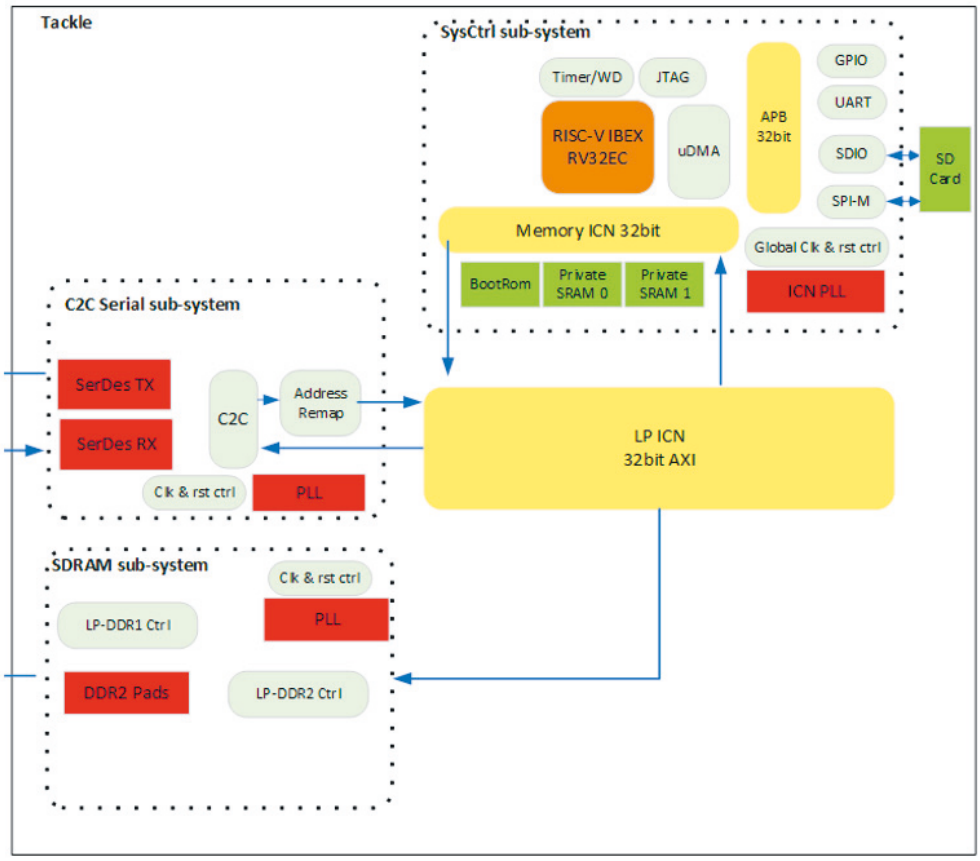

Figure 5.3 High-level block diagram of the Tackle IP prototype SoC.

- DMA: Highly configurable DMA. Resides inside interconnect (IC). Not visible in the Fig. 5.3 (new IP).
- LP-DDR1/2: Designed by one of the collaborating companies (new IP).
- C2C with SerDes: High-speed serial Transmitter/Receiver (TX/RX) IP for off-chip communication (new IP).
- Interconnect sub-system: single crossbar AXI interconnect, DMA, and Shared Static Random Access Memory (SRAM). fmax 533MHz, area 0,9mm^2.

Headsail is an autonomous Linux-capable SoC that reuses multiple IPs from the Ballast and Tackle but also includes new IPs. For the Headsail custom Ball Grid Array (BGA)-624 pin IC package was designed which eventually allowed the inclusion of multiple interface IPs. The Headsail is a 25mm^2 in size and includes 340M transistors.

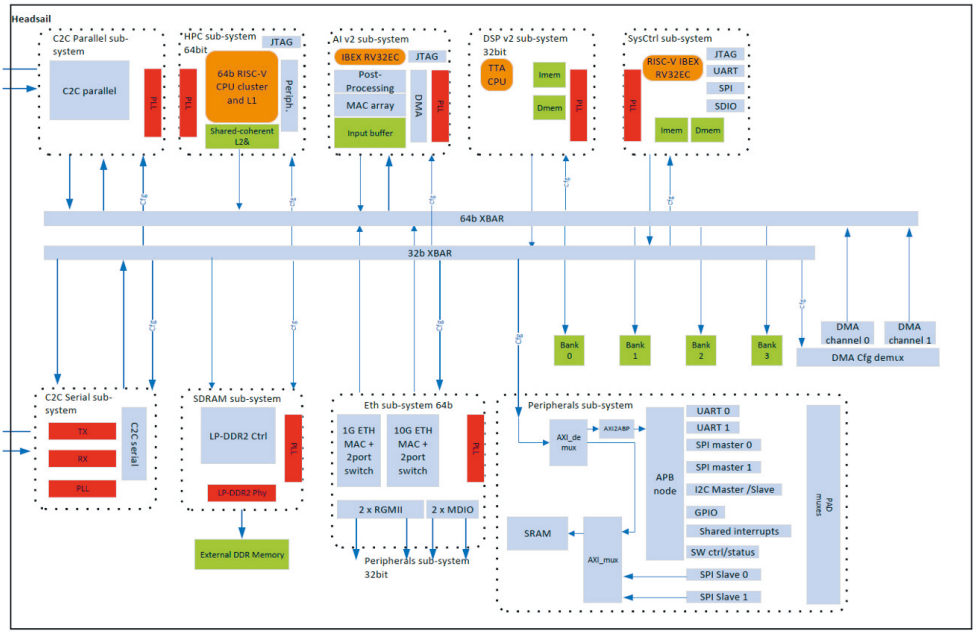

Figure 5.4 High-level block diagram of the Headsail SoC.

The summary of Headsail sub-systems is following:

- PLL: Reused as a hard macro.
- HPC sub-system: Resued from Ballast except the amount of CPU cores was increased from two to four. (reused). fmax 1GHz, area 7mm^2.
- SysCtrl sub-system: Same as in Ballast (reused). fmax 30MHz, area 0,35mm^2.
- AI sub-system: DLA. (new IP). fmax 900Mhz, area 9,4mm^2
- DSP sub-system: Update from the Ballast version [37] (partly reused). fmax 1GHz, area 0,74mm^2
- C2C parallel sub-system: Direct Reuse from the Ballast (reused). fmax 400MHz, Ios 100MHz, area 1mm^2.
- C2C serial sub-system: Chip-to-chip with SerDes mixed signal macros. Major update from the Tackle version (new IP). fmax 3GHz, area 0,4mm^2 excl. analog macros.
- Eth sub-system: Dual port 1G IP reused from collaboration company, but ASIC physical design was new. (reused/new IP). fmax 125MHz, area 1,5mm^2.

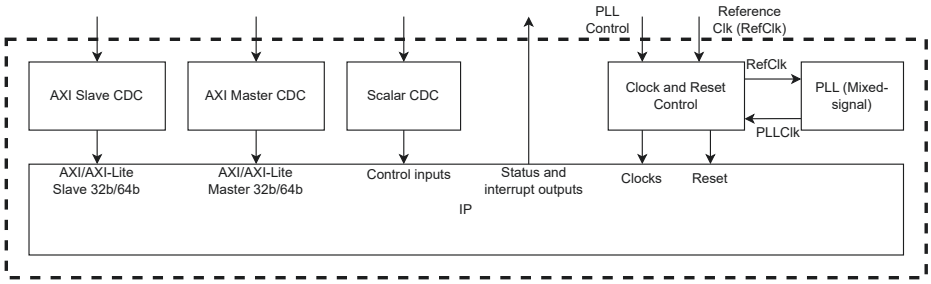

Figure 5.5 Template design for sub-systems.

- peripheral sub-system: New sub-system, but components where reused from pulp-platform repositories (reused/New IP). fmax 125MHz, part of the top level area.
- Interconnect sub-system: Two stage-crossbar interconnect, DMA, and Shared SRAM. fmax 1Ghz, area 3,8mm^2.

5.3 Chip design

The chip architectures were hierarchically constructed from sub-systems that included the lowest granularity level IP blocks. We employed a template-based design approach to encapsulate clock source (PLL), data and control interfaces, clock domain boundaries, and reset signals for each subsystem as seen in Fig. 5.5[70]. All subsystems were developed concurrently from RTL to the Design Rule Check (DRC)-cleaned GDSII layout, and the approach scaled well independent of the complexity and number of subsystems. This approach reduced the overall physical design time, simplified integration, and reduced risks for interfacing bugs.

The design automation started at the modeling phase. We used the IP-XACT standard and Kactus2 [85] tool to model the architecture and memory maps. Open-source IPs were modeled at least to the subsystem interface level, and the top level and the new IPs in the most detail. Some open-source IPs require extensive IP-XACT modeling effort, and balanced the time spent and the level of detail [63]. To show the scale, Headsail included 226 IP-XACT documents. We then applied extensively RTL code generation from the IP-XACT models, resulting in the order of a hundred thousand lines of code. This was repeated multiple times because architectural changes took place even quite late in the schedule. We also used the Keelhaul [55] tool for

validating the memory maps. The rest of the automation occurred in a specific CI pipeline and commercial EDA tools.

6 WHAT AGILE METHODS OF SOC DESIGN LOOK LIKE?

When forming a definition for the Agile HW Development methodology, multiple aspects need to be considered. One needs to define a new mindset to get rid of old, in this case, waterfall-minded thinking. In addition to this, practical guidance is needed on how to organize the work and which kind of technical approach agile development benefits from. Based on our research [67] we define Agile HW Development with five instructive values. Three of them originated from the work by Lee et al. [53] and one new values updates Agile HW Development methodology. To recap, defining values are the following:

1) Predictable schedule and fixed cadence., new value.

2) Incomplete, fabricable prototypes over fully featured models. [53]

3) Collaborative, flexible teams over rigid silos. [53]

4) Interface over instance. update from *Improvement of tools and generators over the improvement of the instance [53].*

5) Response to change over following a plan. [53]

6.1 Predictable schedule and fixed cadence.

The *first time right* approach does not work. First time right assumes that a complex SoC can be finalized from requirements to a production quality level chip with a single design iteration. Based on a large survey [2], 70% of ASIC projects need two or more project spins (iterations) before production. As a mindset, one needs to admit that building a complex SoCs needs multiple chip development iterations. Traditionally, re-spins have been done due to bugs in design. The change in mindset is that still one product needs multiple re-spins, but those are scheduled beforehand to increase predictability and each iteration adds more features to the product.

Predictability is made possible with the *schedule over content* approach. This

approach is illustrated in Fig. 6.1. Requirements (req) 1, 2, and 4 are feasible to implement in ongoing chip iteration. When time passes, the probability to take requirements in decreases and reqs 3 and 6 are pushed directly to the next chip iteration. Req 5 is started in the current chip iteration, but eventually turns out not to be feasible to implement, and it is pushed to the next chip iteration.

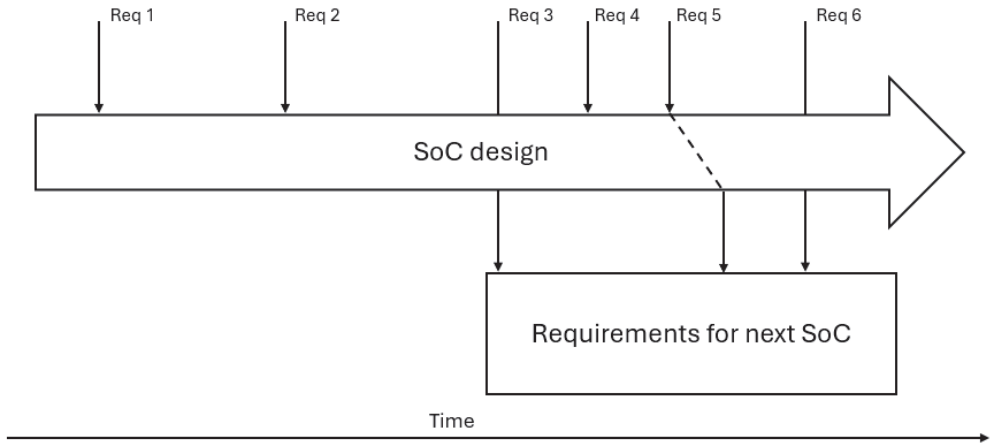

Figure 6.1 Schedule over content approach.

Our research [67] with SoC-Hub indicates that a fixed cadence with an ambitious schedule is possible.

Putting this value in action is challenging. To be able to implement *a schedule over content* approach, there needs to be a way to determine that some of the IPs are mature enough so that it can be included to design. The Milestones introduced in Chapter 3 are the proposed solution for this. Milestone is a measure of the quality and completeness, which is used to guide project decisions and evaluate risks. To avoid this process to delay schedule, the evaluation of the quality is made along the project. If some IP is not feature ready or lacks verification, it is still included with the best effort. This guarantees earlier feedback from full design flow, compared to the case where IP would be pushed completely to the next project round. The other challenge is that how to define the length of the cadence. Most likely, the experience-based estimation is in common use and that was used in the SoC-Hub as well. Dependencies to external vendors also affect the schedule. A common understanding is that projects can be completed in 9 to 18 months. As an example, Apple has released a new iPhone model almost every year. Too short cadence would lead to too small feature

increments and/or increase risks and correspondingly, and too long cadence would lead to too slow iteration loop.

6.2 Incomplete, fabricable prototypes over fully featured models

Above, we concentrated on the feedback loop happening around chip iterations. From the agility point of view, iterations should also happen during the development. Iterations should be fast-paced to enable frequent and early feedback. To improve turn-a-round time and RTL development productivity, methodologies studied in the Chapter 4 are proposed by the literature. One challenge for fast iterations in HW development is long tool runtimes. That may cause, at least for bigger designs, that it is beneficial to group multiple feature updates to one delivery. This process is depicted is the Fig. 6.2. Despite the release cycle, the regression testing should be automated with CI pipelines.

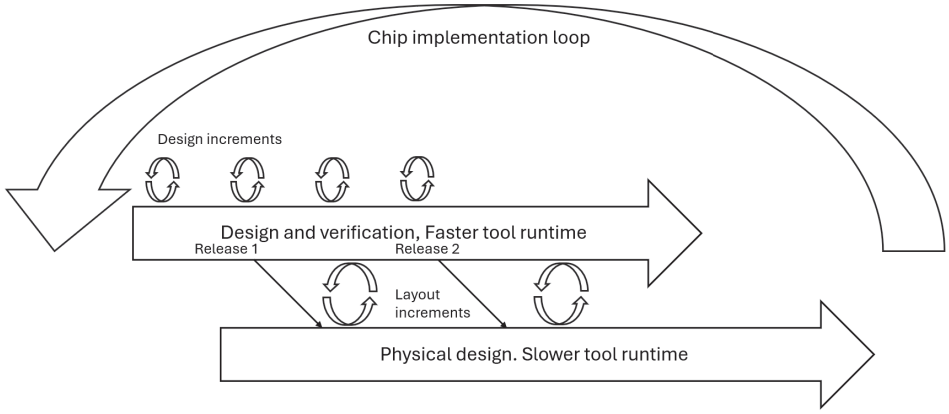

Figure 6.2 Iteration loops of the SoC design.

An efficient method also for faster feedback is to start exploration early with RTL models instead of over-engineering high-level models. The RTL models are more accurate in Power Performance Area (PPA) compared to high-level models. However, one needs to be careful with early models when verification and physical design is run in parallel and also in an incremental fashion. Physical design can benefit from the incomplete, non-functional design only if the increments provide reasonable PPA estimations. Sudden changes in IOs, clock tree, and area of the design can lead to a major replanning of the layout and immature RTL can lead to tedious debug. This

means that increments must be planned and communicated well within the project. Milestones can also be used as a release points to synchronize the work between activities.

We tackled the incompleteness by sub-system template-based architecture design. Each subsystem evolved in parallel with frequent layout runs, which contributed to the top-level floorplanning. With unified architecture, we could also create multiple different design configurations for simulation and FPGA prototyping. We avoided some over-modeling and decided to keep the functional development at RTL abstraction instead of high-level functional models. High-level models can still be used when modeling libraries and model templates are available, but preferably not from scratch.

6.3 Collaborative, flexible teams over rigid silos

In addition to different technical approaches, attention must also be placed on the organization structure, communication, collaboration, competence growth, and team spirit in general. From a communication perspective, the most efficient organization has a flat hierarchy, and teams or even individuals are multidisciplinary. The information does not need to cross organizational structures, and communication of interdisciplinary topics becomes easier when engineers are working (or have been working) in different design activities. Rotation of the tasks is also an effective way to increase competence. Multidisciplinary teams have been proven to work for chips on a moderate scale [53], [23], [67].

To illustrate project organization with multidisciplinary teams let's use the Ballast HW team excluding FPGA, project management, and IT as an example and show the use of persons in Table 6.1 in design, verification, and physical design activities. Ballast included 16 individuals marked R1-R16. As seen, a single person participated in multiple activities in a sub-system.

Some activities did not involve multidisciplinary work. The top-level physical design was dedicated to a single person because of the special expertise required. In addition to sub-systems teams, there were dedicated teams for the FPGA prototyping. The project was led by a single person with an SoC engineering background because they can also talk on design topics rather than inform management about what has been done. There was also a dedicated person for maintaining the EDA tool infrastructure.

Table 6.1 Ballast project organization.

Sub-system	Design	Verification	Physical design	mult.-disciplined team
HPC	R1,R11	R1	R2	Yes
MPC	R3	R3, R13	R11	Yes
SysCtrl	R3, R14	R2, R13, R14	R11	Yes
C2C	R5	R4	R5	Yes
AI	R6	R6	R7	Yes
Eth	R8	R8	R12	Yes
DSP	R9	R15	R9	Yes
Top	R10	R16	R11	No

Having as flat an organization hierarchy as possible was used to reduce communication overhead. Sub-system teams were small (2-4 persons) and they handled the topics directly between the developers and in the general project meetings. We noticed that four persons in one team started to be a limit where dedicated meetings make sense.

SoC-Hub developers had varying experience and competence levels, but together they formed by nature a multidisciplinary team. The team size and skill set were estimated from industry projects. We succeeded in moving the same persons to different design activities. They also reported it as a great learning experience. Another success was the management of a large turnover of the persons between the chips. On-boarding new members was eased because of the collaboration environment, template thinking, and keeping critical senior persons available at least for consultancy. One success factor is the open culture for communication and knowledge sharing.

6.4 Interface over instance

Although the literature proposes multiple different development methods to increase development efficiency, such as MDD, HLS and HCLs [68], there resides some challenges. The problem is that the high-level models themselves become complex because the low-level functionality of the HW is difficult to abstract, i.e. the increase

in abstraction level is moderate. HLS methodology is limited to datapath-oriented designs, and HCLs are still in the early phase and lack literature, training, and design guidelines. Time will show how these and the recently proposed AI-based proposals come to fruition. Meanwhile, the agile HW development community should concentrate on approaches independent of design methodology.

Based on SoC-Hub experience, template-based design with harmonized interfaces is beneficial for multiple reasons. The template forms a "Correct-by-Construct" approach, which reduces risks for bugs. When the sub-system template includes a complete clock and reset tree, physical designs can be run hierarchically so that different sub-systems are placed and routed independently and thus in parallel. This approach can be used regardless of the development methodology. Lee et al. has value *Improvement of tools and generators over the improvement of the instance*, but we propose to replace it with value *Interface over instance*. Unified interfaces also make potential generators more feasible because homogeneous structures are typically low-hanging fruit for code generation.

For modeling, SoC-Hub primarily utilized the Kactus2 IP-XACT tool, which underwent further development during the three SoC designs. However, the IP-XACT standard addresses only structure, for which reason functionality of the IP blocks was included in the RTL code. This is different compared to Chisel generators. IP-XACT fits better with our IP component integration approach. In addition, reuse at physical design alleviates the interfaces rather than the instance functionality. We dare to rephrase this Agile HW Development value as "interface over instance" no matter how the functionality is realized.

6.5 Response to change over following a plan

Responsiveness is a common adjective to describe agile development and both, Agile SW Development and Agile HW Development [53] proposes value *Response to change over following a plan*. In the waterfall persuasion, requirements must be frozen in the early phase so that the project becomes predictable. The problem, however, is that requirements have a high likelihood to change. Customers could request changes to requirements, or development could reveal surprises about the feasibility along the way. For example, IP development (internal or external) might take longer than expected or design might result in too high power consumption. In other words, the

future is unknown. To be able to remove unknowns as early as possible, one should start with the information available and welcome new requirements. They can be implemented in forthcoming chips if new requirements turn out not to be feasible to implement in the ongoing project. However, there are differences and limitations to how this can be realized.

In SoC HW development, the possibility to take new requirements in reduces more near the project is to tapeout and after a certain point there is no other option than pushing implementation of the new requirement to the next chip iteration. In contrast, SW is more elastic and can be updated virtually for free. Again, *schedule over content* approach and incremental design works as a great tool. If there is a likelihood that a new requirement risks the schedule, it is not taken in into the current project iteration, or it is implemented with the best effort with a reduced feature set. Partial implementation makes sense because the requirement get feedback from the full development cycle and works as a prototype.

In Ballast, we both dropped and added features late in the process. For example, Bluetooth IP was planned but eventually left out due to the development time needed for analog macros. Instead, CVA6-based HPC sub-system and NVDLA were included on the fly. CVA6 and NVDLA were mature enough to start experiments already in the Ballast, even though they were more relevant to later SoCs.

The most significant change in the plan was in Tackle. We adhered to "schedule over content" to maintain the agreed timeline, as extending it would have undermined the fixed cadence. The Tackle was meant to be an IO chip with LP-DDR2, multi-port 1G Ethernet, High-speed SerDes, and C2C with a parallel interface. We abandoned the plan and reduced the number of IPs because the IO-chip would have required a custom BGA or flip-chip package that was not yet available on the schedule. We discovered this when the Tackle design was already in the initial layout phase. Due to package constraints, Tackle was thus turned into a mixed-signal IP prototyping chip with a conventional 88-pin QFP package.

Headsail architecture lived during the first months but did not cause any big turbulence. We had plans to change the CVA6 for better performance, but we decided to keep overall changes manageable. In the early roadmapping phase, the HLS-based Kvazaar [80] HEVC video encoder accelerator was also in our plans, but we pushed that to upcoming chips. We dropped it because of the huge on-chip memory requirement. Unlike the Tackle, Headsail 624-pin BGA IC package design

had no impact on the chip architecture, but demanding package design lead to delays in post-tapeout phase.

7 DEVELOPMENT EFFICIENCY AND QUALITY

Quantified results are provided in this Chapter, to make better conclusions on agility. The development process measurements are common in SW engineering and provide support for better understanding, evaluation, and control of the development process, project, and the resulting product [58]. Common Key Performance Indicators (KPI) for the development phase are time, content, cost, and quality [7]. In this thesis, we concentrate on time, costs (personnel) and quality aspects.

This chapter answers research questions RQ3 (How much resources are needed when SoCs are built with Agile HW Development methodologies?), and RQ4 (How the development process quality can be measured?)

7.1 Related work

When we talk about development methodology efficiency, some quantified results should be shared. Table 7.4 summarizes the state of the art in the form of the size of the engineering teams and time spent. Short descriptions about the chip are also reported to put results in scale. The presence of an agile approach is also included. In summary, SoC projects take 3 to 9 months to complete, and the typical team size is 10–20 persons. The size of the teams indicate that projects are rather small scale and include a lot of reuse. Even if quantified results are reported, none of the related work accurately specifies the measurement setup, e.g. which design phases are included in the results.

Table 7.1 Team size and Project duration of selected SoCs 1/2.

Publication	Team size	Time spent	Description	Agile approach
[97]	5 undergraduates	-	64bit RISC-V processor.	No
[53]	-	11 chips in 5 years	RISC-V processor prototypes.	Yes
[39]	<10	4 months	SoC integration project.	Yes
[9]	-	14 000 engineering hours.	A mixed-signal RISC-V SoC.	Yes
[23]	-	9 months	511 core RISC-V platform.	No
[34]	19 students	14 weeks	A Heterogeneous SoC for Bluetooth LE in 28nm technology.	No
[30]	32	15 weeks	A Heterogeneous RISC-V SoC for Machine Learning (ML) Applications in Intel 16nm technology.	No
[49]	-	13 months	Arm Neoverse N2, an industrial CPU for datacenter and edge computing.	No

Table 7.2 Team size and Project duration of selected SoCs 2/2.

Publication	Team size	Time spent	Description	Agile approach
[20]	-	8000 engineering years	NVIDIA SoC for autonomous machines.	No
[52]	7 (4 PhDs, 3 post-docs)	9 months	SoC for IoT with arm Cortex-A53, embedded Field-Programmable Gate Array (eFPGA), and coherent accelerators.	No
[76]	3 full-time, 4 part-time	7 months	Eight-core RISC-V vector processor SoC	Yes
[94]	-	12 weeks	Multi-processor RISC-V SoC	Yes

Table 7.3 SoC hub project milestones.

Milestone	description
HW0	Kick-off
HW1	Architecture and project setup
HW2	Early RTL
HW3	Sub-systems RTL verified
HW4	Top RTL verified
HW5	Sub-system GDSII
HW6	Top-level GDSII
HW7	Tapeout
HW8	Sample testing
HW9	Applications

7.2 Development time and resources

The development time from the requirements to SoC tapeout is a simple KPI measure. One important note, however, is that when a project uses an agile approach, development time needs to be understood a bit differently for SoC-Hub use-case. This is because the project development time is set beforehand and the "schedule over content" approach reduces the risks of project delays. In this situation, the development time measures how well a project was able to adapt to changing situations, i.e. how agile the project was. While the project used milestones to plan and track the project, we can provide insight on which phase the project schedule faced challenges. As a recap, milestones used in our projects are summarized in the Table 7.3

Year	2020												2021												2022												2023					
Month	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6
Ballast: Planned			HW0			HW1				HW2				HW3		HW4		HW5							HW6	HW7																	
Ballast: Realized			HW0			HW1				HW2				HW3				HW4			HW5	HW6	HW7																			
Tackle: Planned																									HW0		HW1		HW2	HW3	HW4		HW5	HW6	HW7							
Tackle: Realized																									HW0		HW1		HW1	HW2	HW3	HW4	HW5	HW6	HW7							
Headsail: Planned																																	HW0			HW1		HW2	HW3		HW4	HW5
Headsail: Realized																																	HW0			HW1		HW2		HW3	HW4	HW5

Figure 7.1 Planned and actual project schedules for the three chips. Delays are highlighted in yellow.

The target schedule was one year from the specification to the tapeout. Milestones were evenly distributed along the timeline, as depicted in Figure 7.1. *HW0-HW7* are represented for the planned and actual timing. Months highlighted in yellow show notable delays compared to the plan. The Ballast spent remarkably time setting up the environments between HW0-HW2. Since this effort is non-recurring, it was excluded from the project duration discussed later.

We encountered two notable challenges but managed to recover successfully. Ballast memory models were received late, which caused a delay in Ballast HW4 and HW5. Tackle required a major architectural change due to pin limitations in the IC package, shown as a redo of HW1 at P5 in 2022. For the Headsail, we combined separate subsystem and top-level milestones. Having too many will not fully reflect the parallel and agile nature of the development.

Verification across all projects took longer than expected, but verification of the corner cases was run in parallel with the physical design. Despite the challenges, none of them caused delays to tapeouts, and we were able to deliver three functional SoCs with the annual cadence. As a takeaway, we learned that 3-6 months need to be reserved to make contracts in a cold-start with the technology providers, and IC package and PCB designs must be started at the same time with the architectural chip planning.

Projects durations were 12 months for Ballast, 10 months for Tackle and 9 months for Headsail. Two later ones deviate from the one-year target, but we had to reduce verification effort at know risks due to the resourcing issues.

The number of person months spent and the number of personnel are presented in Table 7.4. These together with Information Technology (IT) represent the development costs. Tool license costs are excluded because this is confidential information. The use of resources is split per design activity. In our project organization, we assigned both design and verification tasks mostly to the same person. Consequently, the results for Design and Verification (D&V) are combined. Reported results are normalized by the level of experience. Used scaling factors were Junior (0,5), Senior (0,75), and Expert (1,0). Junior means master's graduate or trainee, senior has at least three years and expert more than ten years experience. The average experience level was around 0,75 for all projects.

Three new IPs were developed: Two C2C with parallel and SerDes interfaces, and DLA. Their development effort is reported separately. PLL, Headsail multi-port

1G Ethernet, and LP-DDR2 IPs were designed by the collaborating company. PLL was delivered as a GDSII and its design effort is left out of this analysis. Ethernet and LP-DDR2 were delivered in RTL and its physical design and integration verification are included in the analysis.

Ballast was our first project and included an extensive environment setup. This is seen in Ballast FPGA and IT infrastructure results, including simulation setups. The drawback of open-source IP is that a lot of reverse engineering is needed, even for small modifications. In total, Ballast took 132,5 scaled person months to complete.

The design effort for the Tackle was remarkably smaller (79,2 person months) due to reuse and smaller chip area. C2C SerDes took close to the same amount of time compared to the original C2C with a parallel interface. The uppermost protocol layers for C2C TX and RX were reused from the C2C parallel version, and the majority of the work consisted of analog design. As we see, SoC D&V did not scale down as much as expected. This is because a lot of rework was needed due to changes in the chip architecture. Physical design effort scales down well with the number of sub-systems included. The FPGA prototyping flow was reused from the Ballast.

The Headsail project was the biggest out of the three. SoC D&V is relatively small compared to Ballast and Tackle, which is explained by the high reuse rate and the team know-how. The AI DLA IP was our largest new IP development activity for Headsail and it alone took a significant amount of effort. The physical design took most of the effort because the Headsail had the largest number of sub-systems and the physical design can not be reused similar to design and verification due to layout changes. The reason for the low value for FPGA work is that we just did not have enough human resources. RTL verification was prioritized over FPGA prototyping, and only SoC boot was validated with FPGA. That situation was not optimal, but we took the risk. Headsail was very successful, and it took 121,3 person months were spent.

When compared to the related work, SoC-Hub projects took longer and had larger team. The related work does not provide enough details to explain the reasons, but potentially staff turnover and mentoring of juniors increased the design time. Also, projects listed in related work might be more of integration projects with less IP development. On the other hand, our SoC development flow was comparatively lightweight when measured against the comprehensive methodologies typically employed in industry-grade development. The IP verification process did not achieve full

coverage, and power domains were not implemented to support low-power operation.

Table 7.4 Amount of the person months per design activity. abbreviations: Design (D), Verification (V) & Analog (A).

Activity / Project	Ballast		Tackle		Headsail	
Resources:	Person months	%	Person months	%	Person months	%
D&V	42,8	32%	31,9	44%	34,0	28%
D & V, (C2C IP)	16,3	12 %	-	-	-	-
D, V & A (C2C SerDes 1.0 IP)	-	-	17,1	23%	-	-
D, V & A (C2C SerDes 2.0 IP)	-	-	-	-	9,2	8%
D & V, (DLA IP)	-	-	-	-	20,6	17%
Physical design	29,5	22%	17,6	24%	48,1	40%
FPGA	27,8	21%	0,5	1%	0,8	1%
IT Infra	5,6	4%	1,0	1%	1,0	1%
Management	10,6	8%	4,8	7%	7,5	6%
Total person months	132,5	100%	79,2	100%	121,3	100%
Personnel	22	-	18	-	24	-
Experience level average	0,73	-	0,81	-	0,78	-

7.3 Development process quality

Although process performance measurements are common in the field of SW engineering, according to our knowledge, results of such measurements have not been published earlier in the SoC context. To improve the situation, we measured our SoC development process quality for the Ballast project [69] with FST methodology [21]. The target is to measure the process quality and at a the same time find out how well FST fits to agile SoC development.

The FST methodology [21] was developed to identify SW test process improvement needs. It is an improvement of earlier fault analysis methodology which tries to capture the root causes for faults, which are then classified to find common denominators. Unfortunately, fault analysis is laborious for large projects and does not provide direct feedback to improve the development process. Therefore, FST methodology classifies faults based on their temporal presence and measures how many **faults slipped through from the cost-efficient test phase** where they "belong to". In addition, the FST methodology estimates the improvement potential of the different test phases. The primary result is the improvement potential in hours. For that, one must also define the **average fault cost per phase**, which allows comparing the fix costs in later phases compared to an earlier phase where it should have been found. Our cost estimates are based on interviews with the project members, which is one of the recommended methods by Damm et al. [21].

Fig. 7.2 illustrates the concept with three development phases. Light dots denote the cost of various fixes, and the average cost rises steadily as the system gets larger and more mature (more dependencies, longer tool runtimes). Some bugs "belong to" a certain phase, e.g. wire congestion is visible in physical design (PD) and very unlikely to be fixed in RTL or verification steps. Since activities overlap, we still find errors with simulation during the PD step, although that is not a PD activity *per se*. This **main idea of FST is shown with triangles and dark dots**, namely bugs that could have been found and fixed earlier but were not. Delayed fixes cause extra costs and can be interpreted as improvement potential of the process. Slow multistep SoC Development process and tool flow (Described in section 3.1), long tool runtimes, and high monetary costs of fabrication causes costs of slipped bugs to be high for SoC designs. This situation greatly justifies the usage of process quality methodologies such as FST.

Table 7.5 Git commits per sub-system in Ballast SoC.

Sub-system	Number of commits		
	<HW2	HW2-7	Incl. to FST
HPC	1 846	90	25
MPC	393	199	46
SysCtrl	493	182	71
C2C	110	39	17
AI	522	10	5
Eth	92	70	28
DSP	101	146	31
Interconnect	28	25	18
Total	3 585	1 681	241

7.3.1 Repository workflow and repository-based data

In this work, we collected fault data from commits to the Git version control repository. The project did not use the GitLab *issue* feature to report findings, which would have been the most accurate source for issues. Nevertheless, our robust Git workflow guidelines encouraged frequent commits and teams used descriptive commit messages which typically pointed to a fix of a single issue, such as *"L2 Cache controlled bypass fix"*. Such small commits are also rather fast-paced, so temporal accuracy is on the desired level for this study. The repository layout was hierarchical and matched the structure of the design. Each sub-system had its Git repository, and the repositories were bound together with an integration-level repository.

7.3.2 Analysis of the repository data

The process for collecting data included analysis of the project repositories with the Python repository mining application named PyDriller [82]. An important decision for our FST measurements was that we started collecting the data from the HW2 milestone onwards. This is because work before that is mainly architectural sketching and verification has not started yet.

Eight repositories included together 3585 commits before HW2. The C2C, Ethernet, and DSP sub-systems and L2 cache and boot control modifications were

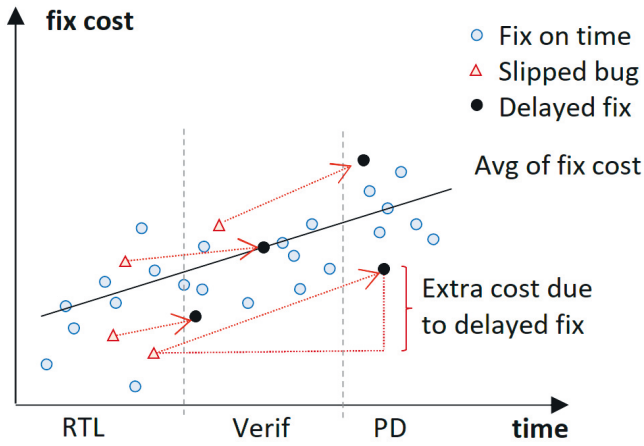

Figure 7.2 Cost of bugs curve [69].

started before the SoC project as individual IP development projects. At the same time, HPC, MPC, SysCtrl, and AI reused large entities from the public repositories. Teams made 1681 commits between HW2-HW7 and we studied those. The actual dataset for FST analysis (belonging vs slipping) considered only RTL design changes, whereas verification environment-related commits were excluded. This led to 241 commits analyzed in detail.

7.3.3 Average fault costs

Table 7.6 The average fault costs per milestone.

Fault cost (h)	Milestone					
	HW3	HW4	HW5	HW6	HW7	HW8
Avg	14	33	49	75	90	570 (fab time. 480 + 90)
Range	1-36	10-72	25-80	40-100	80 - 100	560 - 580

The fault cost per milestone was collected with an email questionnaire. Questions were sent to the project team of 24 persons plus one industry expert outside the project organization. Four answers were received (two seniors and two experts, altogether owning 50 years of cumulative expertise). The response rate was low potentially due to significant number of junior developers. It might be that junior developers felt that they do not have enough experience to reliably evaluate the costs. Responders were guided to evaluate fault costs per milestone for the SoC project

with complexity similar to Ballast. Also, it was guided to include whole iteration time to fault costs including debugging, development of new tests, actual fixes, and tool runtimes. Because one target of this work is to showcase the applicability of the FST to SoC development, the results based on the data are still regarded as valuable despite the low response rate. Estimated fault costs are presented in Table 7.6 and Fig. 7.3. Fabrication time used was three months, which makes 480 working hours with 40 hours of working time per week.

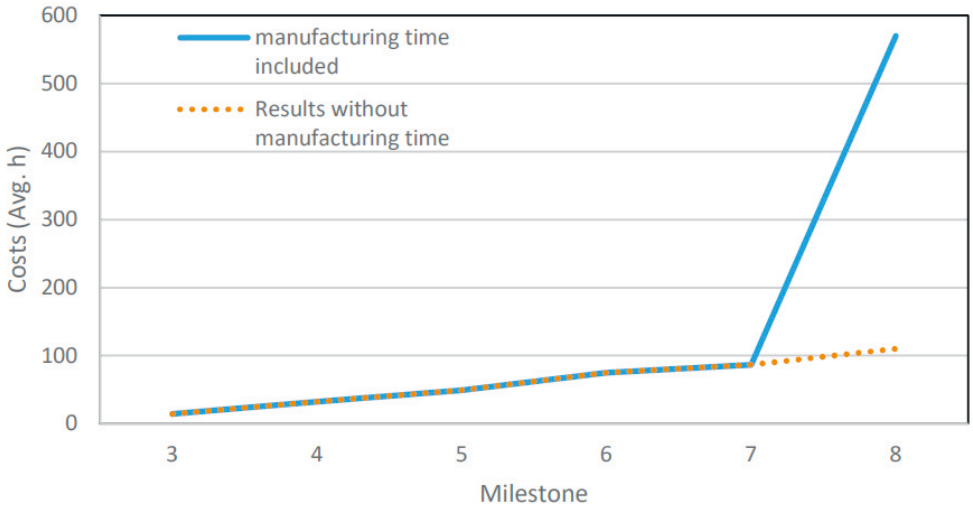

Figure 7.3 Average fault costs.

The increase in the costs is linear between milestones HW3-HW7. The increase could be explained by the increased tool runtimes presented in section 3.1 and maybe debugging time did not cause too long delays due to template-based design architecture. Based on the author's experience, in large SoC industry-scale SoC projects, the penalty is more toward an exponential curve. This is not only due to even longer runtimes, but also increased difficulty of the debug. In large projects, it is very difficult to trace HW behavior along multiple sub-systems. Representatives from multiple teams are needed and time spent on communication starts to account. In our case, the flatter organization most likely helped to overcome the described issue. Iteration time increases by a giant leap after tapeout milestone HW7. Milestone HW8 includes the time needed for fabrication, which depends on varying silicon fab schedules and logistics, here we used factory time of 3 months as an example.

Any faults detected upon sample testing suffer from the limited visibility of the

chip's internal functionality. This makes debugging time-consuming. Even more costly is a new tapeout which is typically required if a SW-based workaround is not possible. Hence, there is a severe time penalty and also a monetary one due to the re-spin. SW development does not encounter these problems. The monetary costs of the new tapeout are excluded from the presented analysis, while information is behind non-disclosure agreements.

During the interviews, we noticed that defining the slip costs is a difficult task and some of the interviewed persons skipped the cost evaluation for some of the milestones. This might indicate that the penalties are visible in other activities later. For example, an RTL design and verification engineer does not directly see the cost of additional physical design iterations. Also, when a fix is known, there is often a temptation to estimate its costs alone but exclude some of the failed attempts. Note also that there are major differences in estimated costs, especially in the first milestones, although estimates for later ones do converge. Despite the potential inaccuracies, these estimates can be used to increase people's awareness about the bug costs.

7.3.4 Fault-Slip-Through results

After collecting the raw data from the repositories, the data was analyzed and filtered. The relevant commits were placed on a correct milestone based on the commit date, and inclusion was done based on the commit message. The FST data is presented in Table 7.7.

Table 7.7 Collected FST data: number of faults per phase.

Belongs to:	When fixed						Total	
	Sub-sys. RTL	Top RTL	Sub-sys. GDSII	Top GDSII	Tapeout	Sample testing	#faults	#Slips
Sub-sys. RTL	118	46	17	1	0	1	183	65
Top RTL	-	39	3	0	0	1	43	4
Sub-sys. GDSII	-	-	13	0	0	0	13	0
Top GDSII	-	-	-	0	0	0	0	0
Tapeout	-	-	-	-	0	0	0	0
Sample testing	-	-	-	-	-	0	0	0
Total #faults	118	85	33	1	0	2	239	69
Total fault cost [h]	1700	2800	1600	100	0	1100	7300	-
Total slip costs [h]	0	1500	1100	100	0	1100	–	3700
Slip cost [%]	0%	54%	61%	100%	0%	100%	0	51%

During the data analysis, we noticed that commits were frequent and included descriptive commit messages. The quality of the commits did not vary between subsystems significantly. We noticed that deciding to which milestone each fix commit belongs was relatively straightforward, and only in a few cases was an inspection of the committed code needed. These findings indicate that repository data is a suitable dataset for FST methodology and repository mining tools help in analyzing data accurately and in a fast manner.

The results give interesting and important insight into the project as a retrospective. A total of 239 bugs were discovered and fixed, but 69 slipped and caused extra costs. The bottom rows show the cost of bug fixing using the estimated average cost. Fault slips account for only 28% of the bug count, but 51% of the time needed to fix them. Regarding the slipped bugs, the reading of the Table 7.8 goes so that everything above diagonal 118, 39, 13, 0, 0, 0 are findings that slipped through the process. The total fault cost values in hours are *number of bugs * average fault costs (Table 7.6*) and fault cost results are rounded to the nearest hundred.

The verification work before *HW3 sub-system RTL verified* milestone catches the largest number of bugs (#118, 49%). None of them are counted as slipped because this is the point where those bugs should have been found. This indicates that verification was started in the planned project phase. The next column with *HW4 top-level RTL verified* data reveals that we continued to find 46 sub-system bugs at SoC-level verification, although sub-system verification was quality-checked with the milestone reviews. This indicates that especially the HW4 milestone review could have been more thorough. An important note here is that a large number (17) of faults belonging to HW3 still slipped in HW5 were related to BootROM development, and it was agreed with the physical design team that the development of that instance can be extended over the previous milestone.

The fault slips practically stopped after *HW5 Sub-system GDSII*. The most severe slips in the SoC development are the ones that are not caught before the tapeout. In our case, there were two bugs discovered at milestone *HW8 Sample testing*, neither of them causing the chip to fail. One slipped bug was related to the memory integration and the other was an architectural bug in interrupt handling. The memory bug slipped because last-minute RTL change was simulated only with the abstract model and simulations with the more accurate technology-specific model were forgotten. The other slippage was caused by incomplete specification of interrupts and the CVA6

CPU was not able to handle UART interrupts.

As a summary, most of the bugs were found at the verification-focused milestones. Despite there being slipped bugs, debugging and solving them was fluent. This is in line with the presented average fault costs for projects in our scale where the increase in costs is not so drastic. However, here we need to remember that the costs of the bugs are way higher with chips on an industry scale. Although there were slipped bugs, the project did not suffer major delays in schedule. Good results on our project success are amplified by the fact that the SoC under evaluation was our first chip in the roadmap of multiple SoCs. The bugs slipped into the fabricated chip are expensive. Debugging is difficult and fixing can cause a lot of rework, and an extra round of fabrication takes time. The fabrication-related monetary costs are also known to be huge, but those costs are out of the scope of this paper. Two slipped bugs to a fabricated chip is not plenty, but it gives good lessons to learn for the next project.

7.3.5 Improvement potential

Table 7.8 Improvement potential.

Belongs to:	When fixed						Total
	Sub-sys. RTL	Top RTL	Sub-sys. GDSII	Top GDSII	Tapeout	Sample testing	[hours]
Sub-sys. RTL	0	-840	-595	-61	0	-556	-2051
Top RTL	-	0	-50	0	0	0	-538
Sub-sys. GDSII	-	-	0	0	0	0	0
Top GDSII	-	-	-	0	0	0	0
Tapeout	-	-	-	-	0	0	0
Sample testing	-	-	-	-	-	0	0
Change in cost [h]	0	-840	-645	-61	0	-1093	-2639 hours -330 person days

Table 7.8 presents the calculated improvement potential. The total improvement potential is about 2600 hours. In other words, if project execution had been ideal without any bugs slipping, we could have ideally saved 330 person days of "cost". That seems like a realistic value taking the complexity of the project into account: the total effort was 133 person-months or approx. 21k person-hours. Hence, the ideal saving could be approx. 2-5%, but the exact number depends on the (currently unknown) breakdown between human time and run time. In other words, saving could be about one full-time equivalent. That does not sound much, but again, if thinking of the monetary side, the money saved could be used for tool licenses, faster prototyping platforms, and faster servers running toolflows which also can greatly accelerate SoC projects.

The relation between the saving in effort and one in the project's schedule is not linear, due to parallel activities, for example. Nevertheless, the data indicates significant improvement potential. Better project practices could have been stricter milestone reviews, code reviews, and coverage analysis. Such actions are important to reach high quality, but everything cannot be left-shifted to the beginning. Otherwise, we would over-engineer and stick the first steps far too long (analysis paralysis) and move on too late. This means that FST may not be the most accurate method for agile projects. Developing process quality measurement methodology for the agile projects is the subject of further research.

One important thing must be kept in mind. The costs of slipped bugs and improvement potential are based on average fault costs formulated at the beginning of the process. For that reason, having as accurate average fault costs as possible is crucial. For that reason, further research should make a much larger survey including at least tens of responses. Another interesting and maybe most efficient method to improve accuracy would be the iterative usage of FST for multiple projects in the roadmap.

7.4 Takeouts

In summary, our results together with the provided analysis indicate the following:

- A Well-defined SoC development process allows the usage of FST methodology to evaluate the process quality.
- Slipped bugs can be discovered from repository meta-data for FST analysis.

- Cost of fixing bugs increases quite linearly as the SoC project proceeds and jumps up on the chip tapeout, but this linear increase without taking fabrication time into account is most likely true only for relatively small designs compared to designs on the industry scale.
- Because the relation between the saving in effort and the project's schedule is not linear, it is not so clear how much the project would have accelerated without slipped bugs. Nevertheless, costs indicate room for improvements, which could be stricter milestone reviews and improvements in tests in general.
- Because the costs of slipped bugs and improvement potential are based on average fault costs per milestone, it is essential to have as accurate as possible estimation of the average fault costs. The accuracy could be improved by having a more extensive survey with more samples and by collecting cost estimates from multiple projects.
- Defining costs of finding bugs for agile projects can be difficult because in some cases the projects would benefit from verification work done later in the process when previous steps are still uncompleted.

8 CONCLUSIONS

We conclude the thesis by summarizing the findings for the research questions.

RQ1: What is the state of the art of the Agile HW Development?

This research strengthens the hypothesis that SoC HW development is so much different from the SW development that known Agile SW Development practices cannot be directly applied. Because the abstraction level of the predominant HW design languages is low, the development cycle takes longer, compared to design languages with higher abstraction level. One essential matter in agile development is fast iteration loops. It can be seen as an enabler for other aspects of agile, such as responsiveness. There seems not to be a single winning methodology, which might indicate that research on Agile HW Development is in early phase. However, the analysis provided by this thesis might help organizations to select the most suitable development methodologies to accelerate SoC design.

The proposals are grouped into the following categories:

- MDD: Abstraction, model transformations, code generation and synthesis.
- Development platforms.
- HW-SW co-design.
- Open-source hardware.
- Large Language Models

There are two important caveats with these proposals. These use of these methodologies does not directly lead to agile development. Secondly, in most of the cases, development efficiency has not been measured quantitatively, which could be a potential topic for further research. Some barriers can also hinder the adoption of the new proposals. Lack of documentation and small development community are typical for technologies in the early phase. One big issue can also be the growing complexity of the development flows. This is because HW design is still based on netlist synthesis

of the RTL description. That means that new development flows extend the existing flows. New intermediate steps and tools can cause extra work, such as creating the models and debugging of the new implementations.

RQ2: What are the agile methods for SoC design?

In this thesis, we have introduced five agile values to increase the agility of the SoC development:

- Predictable schedule and fixed cadence.
- Incomplete, fabricable prototypes over fully featured models.
- Collaborative, flexible teams over rigid silos.
- Interface over instance.
- Response to change over following a plan.

The feasibility of them has been showcased with SoC-Hub use-case. The implementation of the values is restricted by the nature of the SoC development. Also, the scalability of the agile approach to larger design remains as an open question. Despite potential challenges, larger organizations can use these results as background information when they define agile development more suitable for them.

RQ3: How much resources are needed when SoCs are built with Agile HW Development methodologies?

The results indicate that large SoCs in academic scale can be built within a year with a team of 20 developers and more precisely approximately 100 person months are needed. This is a little bit more than reported in related work, but on the other hand, others do not mention clearly enough which design phases were included and how much work included new IP development. When projects get bigger, maybe the most challenging value to implement is *collaborative, flexible teams over rigid silos*. With complex design, there is more than enough to do within each design activity, and there is no room to move to other development areas than the primary one. This leads to larger separate organizations which likely hinder communication. Because measuring the complexity of the designs is difficult, it is hard to say how results on resource needs scale for larger, more complex design. To improve the situation, it would be beneficial if research could introduce complexity metrics for the projects. That could include parameters such as reuse rate, design area, and experience level of

the personnel.

RQ4: How the development process quality can be measured?

This thesis showcased that FST methodology can be used as a process quality metric for SoC designs. The usage of the FST needs that different design phases are defined in the form of milestones. As a result, FST provides values for the amount of found faults, average fault costs, and the improvement potential. FST measurements were done for the Ballast project. 69 bugs were slipped and two of them were found after the tapeout milestone, but neither of them caused the chip to fail. Project members were interviewed to find the costs of finding bugs in the different phases, and those costs ranged from 1 hour to 580 hours. Eventually, the improvement potential of the SoC development process was presented, and the results indicate that the cost of slipped bugs was 330 engineering days.

Because FST has not been earlier applied to SoC development, there is no reference to compare with. When designs get more complex, the average fault costs most likely increase. This is due to difficulties in debugging, longer tool runtimes and inter-team communication. FST measurements were done only for the first project. At the time, the team did not have so much experience compared to the experience they had after completing all three projects. We encountered bugs as well in the second and third chips, and there the experiences revealed that a lot of time (months) is needed to debug issues in the post-silicon phase. That means that average fault costs are likely much higher what they were estimated during the first project. The suitability of the FST to agile projects can be questioned because different design phases are meant to be run in parallel and slipped bug (before fabrication) can even accelerate the project. Despite potential inaccuracies, these results can be used to increase awareness of the process quality with in project organizations, regardless of the project complexity.

Future work

The early age of Agile HW Development research opens multiple fruitful possibilities for future research. The scalability of the Agile HW Development values should be tested with larger designs and organizations. Also, one should perform more measurements on development efficiency and process quality in different types of settings so that more detailed conclusions could be made.

The Agile HW Development should also be tested with projects operating in

security or safety regulated environments. A rigid certification process most likely needs tailoring of the agile development process. HW development related question is that how and when to judge maturity of the included IPs when IPs are developed in an agile manner with continuous improvements. This is especially relevant for the open-source implementations, where agreed metrics for the maturity do not exist.

REFERENCES

[1] Pekka Abrahamsson, Nilay Oza, and Mikko T Siponen. "Agile software development methods: a comparative Review1". In: *Agile software development: Current research and future directions* (2010), pp. 31–59.

[2] Siemenens AG. *Wilson Research Group functional verification study*. https://resources.sw.siemens.com/en-US/white-paper-2020-wilson-research-group-functional-verification-study-ic-asic-fucntional-verification-trend-report/. Accessed: 2024-10-29. 2020.

[3] Muhammad Ovais Ahmad. "Agile Methods and Cyber-Physical Systems Development—A Review with Preliminary Analysis". In: *International Conference on Big Data and Security*. Springer. 2019, pp. 274–285.

[4] Jacob N Allen, Hoda S Abdel-Aty-Zohdy, and Robert L Ewing. "Agile hardware development with rapid hardware definition language". In: *2009 IEEE International Conference on Electro/Information Technology*. IEEE. 2009, pp. 383–388.

[5] Alon Amid et al. "Chipyard: Integrated design, simulation, and implementation framework for custom socs". In: *IEEE Micro* 40.4 (2020), pp. 10–21.

[6] David J Anderson. *Kanban: successful evolutionary change for your technology business*. Blue hole press, 2010.

[7] Ž Antolić. "An example of using key performance indicators for software development process efficiency evaluation". In: *R&D Center Ericsson Nikola Tesla* 6 (2008), pp. 1–6.

[8] Rick Bahr et al. "Creating an agile hardware design flow". In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2020, pp. 1–6.

[9] Steven Bailey et al. “A mixed-signal risc-v signal analysis soc generator with a 16-nm finfet instance”. In: *IEEE Journal of Solid-State Circuits* 54.10 (2019), pp. 2786–2801.

[10] BairesDev. *Methodologies*. 2025. URL: https://tinyurl.com/bairesdevmeth.

[11] Jonathan Balkind et al. “OpenPiton at 5: A nexus for open and agile hardware design”. In: *IEEE Micro* 40.4 (2020), pp. 22–31.

[12] Kent Beck. *Extreme programming explained: embrace change*. Addison-Wesley, 2000.

[13] Kent Beck et al. *The agile manifesto*. https://agilemanifesto.org/. Accessed: 2024-08-12. 2001.

[14] Pradip Bose et al. “Secure and resilient socs for autonomous vehicles”. In: *Proceedings of the 3rd International Workshop on Domain Specific System Architecture (DOSSA)*. 2021, pp. 1–6.

[15] G Bou Ghantous and Asif Gill. “DevOps: Concepts, practices, tools, benefits and challenges”. In: *PACIS2017* (2017).

[16] Jean Bruant et al. “Toward Agile Hardware Designs With Chisel: A Network Use Case”. In: *IEEE Design & Test* 39.1 (2021), pp. 77–84.

[17] Eric Chang et al. “BAG2: A process-portable framework for generator-based AMS circuit design”. In: *2018 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE. 2018, pp. 1–8.

[18] Po-Han Chen, Charles Tsao, and Priyanka Raina. “An Open-Source 4x8 Coarse-Grained Reconfigurable Array Using SkyWater 130 nm Technology and Agile Hardware Design Flow”. In: *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2023, pp. 1–5.

[19] Lucas Cordeiro et al. “A platform-based software design methodology for embedded control systems: An agile toolkit”. In: *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008)*. IEEE. 2008, pp. 408–417.

[20] NVIDIA Corporation. *NVIDIA’S XAVIER SOC*. 2018. URL: https://old.hotchips.org/hc30/1conf/1.12_Nvidia_XavierHotchips2018Final_814.pdf.

[21] Lars-Ola Damm, Lars Lundberg, and Claes Wohlin. “Faults-slip-through—a concept for measuring the efficiency of the test process”. In: *Software Process: Improvement and Practice* 11.1 (2006), pp. 47–59.

[22] Deeksha Dangwal, Georgios Tzimpragos, and Timothy Sherwood. “Agile hardware development and instrumentation with PyRTL”. In: *IEEE Micro* 40.4 (2020), pp. 76–84.

[23] Scott Davidson et al. “The Celerity open-source 511-core RISC-V tiered accelerator fabric: Fast architectures and design methodologies for fast chips”. In: *IEEE Micro* 38.2 (2018), pp. 30–41.

[24] Giovanni De Michell and Rajesh K Gupta. “Hardware/software co-design”. In: *Proceedings of the IEEE* 85.3 (1997), pp. 349–365.

[25] Surafel Demissie et al. “Agile Usage in Embedded Software Development in Safety Critical Domain–A Systematic Review”. In: *International Conference on Software Process Improvement and Capability Determination*. Springer. 2018, pp. 316–326.

[26] Digital.ai. *State of Agile Report*. https://info.digital.ai/rs/981-LQX-968/images/AR-SA-2022-16th-Annual-State-Of-Agile-Report.pdf. Accessed: 2024-11-28. 2022.

[27] Kim Dikert, Maria Paasivaara, and Casper Lassenius. “Challenges and success factors for large-scale agile transformations: A systematic literature review”. In: *Journal of Systems and Software* 119 (2016), pp. 87–108.

[28] Andrew Dobis et al. “Chiselverify: An open-source hardware verification library for chisel and scala”. In: *2021 IEEE Nordic Circuits and Systems Conference (NorCAS)*. IEEE. 2021, pp. 1–7.

[29] Dominic Eschweiler and Volker Lindenstruth. “Test driven development for device drivers and rapid hardware prototyping”. In: *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE. 2015, pp. 1–9.

[30] Yufeng Chi et.al. “A Heterogeneous RISC-V SoC for ML Applications in Intel 16 Technology”. In: *2023 IEEE Hot Chips 35 Symposium (HCS)*. IEEE, 2023, pp. 1–1.

[31] Kathleen Feng et al. “Amber: A 16-nm System-on-Chip With a Coarse-Grained Reconfigurable Array for Flexible Acceleration of Dense Linear Algebra”. In: *IEEE Journal of Solid-State Circuits* (2023).

[32] Fossi Foundation. *LibreCores*. 2023. URL: https://www.fossi-foundation.org/2022/10/19/librecores.

[33] Google. *Protocol Buffer*. https://https://protobuf.dev/. Accessed: 2024-10-29. 2020.

[34] Felicia Guo et al. “A Heterogeneous SoC for Bluetooth LE in 28nm”. In: *2023 IEEE Hot Chips 35 Symposium (HCS)*. IEEE. 2023, pp. 1–11.

[35] Xinfei Guo et al. “Agile-AES: Implementation of configurable AES primitive with agile design approach”. In: *Integration* 85 (2022), pp. 87–96.

[36] Zicheng He et al. “Agile hardware and software co-design for RISC-V-based multi-precision deep learning microprocessor”. In: *Proceedings of the 28th Asia and South Pacific Design Automation Conference*. 2023, pp. 490–495.

[37] Kari Hepola, Joonas Multanen, and Pekka Jääskeläinen. “Energy-Efficient Exposed Datapath Architecture With a RISC-V Instruction Set Mode”. In: *IEEE Transactions on Computers* (2023).

[38] Pekka Jääskeläinen et al. “HW/SW Co-design Toolset for Customization of Exposed Datapath Processors”. In: (2017), pp. 147–164.

[39] Tianyu Jia et al. “A 12nm agile-designed SoC for swarm-based perception with heterogeneous IP blocks, a reconfigurable memory hierarchy, and an 800MHz multi-plane NoC”. In: *ESSCIRC 2022-IEEE 48th European Solid State Circuits Conference (ESSCIRC)*. IEEE. 2022, pp. 269–272.

[40] Shunning Jiang et al. “PyMTL3: A Python framework for open-source hardware modeling, generation, simulation, and verification”. In: *IEEE Micro* 40.4 (2020), pp. 58–66.

[41] Lizy Kurian John. “Agile Hardware Design”. In: *IEEE Annals of the History of Computing* 40.04 (2020), pp. 4–5.

[42] Andrew B Kahng and Tom Spyrou. “The OpenROAD Project: Unleashing Hardware Innovation”. In: *Proc. Government Microcircuit Applications and Critical Technology Conference*. 2021, pp. 1–6.

[43] Matti Kaisti et al. “Agile methods for embedded systems development-a literature review and a mapping study”. In: *EURASIP Journal on Embedded Systems* 2013.1 (2013), p. 15.

[44] Antti Kamppi et al. “Kactus2: A graphical EDA tool built on the IP-XACT standard”. In: *Journal of Open Source Software* 2.13 (2017), p. 151.

[45] Sagar Karandikar et al. “A Hardware Accelerator for Protocol Buffers”. In: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 2021, pp. 462–478.

[46] Staffs Keele et al. *Guidelines for performing systematic literature reviews in software engineering*. Tech. rep. Citeseer, 2007.

[47] Kurt Keutzer et al. “System-level design: Orthogonalization of concerns and platform-based design”. In: *IEEE transactions on computer-aided design of integrated circuits and systems* 19.12 (2000), pp. 1523–1543.

[48] Brucek Khailany. “Accelerating chip design with machine learning”. In: *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*. 2020, pp. 33–33.

[49] Anitha Kona. “CSS N2: Arm Neoverse N2 Platform, Delivered to Partners as a Fully Verified, Customizable Subsystem”. In: *2023 IEEE Hot Chips 35 Symposium (HCS)*. IEEE Computer Society. 2023, pp. 1–23.

[50] Kaisa Könnölä et al. “Agile methods in embedded system development: Multiple-case study of three industrial cases”. In: *Journal of Systems and Software* 118 (2016), pp. 134–150.

[51] Kalhan Koul et al. “Aha: An agile approach to the design of coarse-grained reconfigurable accelerators and compilers”. In: *ACM Transactions on Embedded Computing Systems* 22.2 (2023), pp. 1–34.

[52] Sae Kyu Lee et al. “Smiv: A 16-nm 25-mm^2 soc for iot with arm cortex-a53, efpga, and coherent accelerators”. In: *IEEE Journal of Solid-State Circuits* 57.2 (2021), pp. 639–650.

[53] Yunsup Lee et al. “An agile approach to building RISC-V microprocessors”. In: *IEEE Micro* 36.2 (2016), pp. 8–20.

[54] Tianyang Liu et al. “ChatChisel: Enabling Agile Hardware Design with Large Language Models”. In: *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*. IEEE. 2024, pp. 710–716.

[55] Henri Lunnikivi, Roni Hämäläinen, and Timo D Hämäläinen. “Keelhaul: Processor-Driven Chip Connectivity and Memory Map Metadata Validator for Large Systems-on-Chip”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2024).

[56] Paolo Mantovani et al. “Agile SoC development with open ESP”. In: *Proceedings of the 39th International Conference on Computer-Aided Design*. 2020, pp. 1–9.

[57] Yiqing Mao et al. “CFEACT: A CGRA-based Framework Enabling Agile CNN and Transformer Accelerator Design”. In: *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE. 2024, pp. 213–219.

[58] Ayman Meidan et al. “Measuring software process: a systematic mapping study”. In: *ACM Computing Surveys (CSUR)* 51.3 (2018), pp. 1–32.

[59] Marco Minutoli et al. “SODA: a new synthesis infrastructure for agile hardware design of machine learning accelerators”. In: *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE. 2020, pp. 1–7.

[60] Antti Nurmi et al. “A Resilient System Design to Boot a RISC-V MPSoC”. In: *2022 25th Euromicro Conference on Digital System Design (DSD)*. IEEE. 2022, pp. 232–238.

[61] NVIDIA. *NVIDIA Deep Learning Accelerator (NVDLA)*. 2017. URL: http://nvdla.org/index.html.

[62] Maria Paasivaara. “Adopting SAFe to scale agile in a globally distributed organization”. In: *2017 IEEE 12th International Conference on Global Software Engineering (ICGSE)*. IEEE. 2017, pp. 36–40.

[63] Esko Pekkarinen and Timo D Hämäläinen. “Modeling risc-v processor in ip-xact”. In: *2018 21st Euromicro Conference on Digital System Design (DSD)*. IEEE. 2018, pp. 140–147.

[64] Daniel Petrisko et al. “BlackParrot: An agile open-source RISC-V multicore for accelerator SoCs”. In: *IEEE Micro* 40.4 (2020), pp. 93–102.

[65] ML Petrović and VM Milovanović. “A Chisel Generator of Parameterizable and Runtime Reconfigurable Linear Insertion Streaming Sorters”. In: *2021 IEEE 32nd International Conference on Microelectronics (MIEL)*. IEEE. 2021, pp. 251–254.

[66] Mary Poppendieck and Tom Poppendieck. *Lean software development: An agile toolkit: An agile toolkit*. Addison-Wesley, 2003.

[67] Antti Rautakoura, Timo Hamalainen, and Ari Kulmala. “Three SoCs in three years-How to get agile?” In: *IEEE Micro* (2025).

[68] Antti Rautakoura and Timo Hämäläinen. “Does soc hardware development become agile by saying so: A literature review and mapping study”. In: *ACM transactions on embedded computing systems* 22.3 (2023), pp. 1–27.

[69] Antti Rautakoura, Erno Salminen, and Timo Hämäläinen. “Measuring the SoC development process quality with Fault-slip-Through methodology”. In: *2024 13th Mediterranean Conference on Embedded Computing (MECO)*. IEEE. 2024, pp. 1–8.

[70] Antti Rautakoura et al. “Ballast: Implementation of a large MP-SoC on 22nm ASIC technology”. In: *2022 25th Euromicro Conference on Digital System Design (DSD)*. IEEE. 2022, pp. 276–283.

[71] Antti Rautakoura et al. “Kamel: IP-XACT compatible intermediate meta-model for IP generation”. In: *2020 23rd Euromicro Conference on Digital System Design (DSD)*. IEEE. 2020, pp. 325–331.

[72] Yang Ren et al. “A Customized Quad-Core RISC-V SoC for Energy Control Units”. In: *2024 IEEE 7th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. Vol. 7. IEEE. 2024, pp. 912–916.

[73] Per Runeson et al. *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons, 2012.

[74] Maico Cassel dos Santos et al. “A scalable methodology for agile chip development with open-source hardware components”. In: *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 2022, pp. 1–9.

[75] Pasquale Davide Schiavone et al. “Quentin: an ultra-low-power pulpissimo soc in 22nm fdx”. In: *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*. IEEE. 2018, pp. 1–3.

[76] Colin Schmidt et al. “An eight-core 1.44-GHz RISC-V vector processor in 16-nm FinFET”. In: *IEEE Journal of Solid-State Circuits* 57.1 (2021), pp. 140–152.

[77] Ken Schwaber and Mike Beedle. *Agile software development with Scrum*. Prentice Hall PTR, 2001.

[78] John Shalf, Dan Quinlan, and Curtis Janssen. “Rethinking hardware-software codesign for exascale systems”. In: *Computer* 44.11 (2011), pp. 22–30.

[79] Nitin Sherje. “Enhancing Software Development Efficiency through AI-Powered Code Generation”. In: *Research Journal of Computer Systems and Engineering* 5.1 (2024), pp. 01–12.

[80] Panu Sjövall et al. “High-level synthesis implementation of an embedded real-time HEVC intra encoder on FPGA for media applications”. In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 27.4 (2022), pp. 1–34.

[81] Tyler Sorensen et al. “A simulator and compiler framework for agile hardware-software co-design evaluation and exploration”. In: *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE. 2020, pp. 1–9.

[82] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. “Pydriller: Python framework for mining software repositories”. In: *Proceedings of the 2018 26th ACM Joint meeting on european software engineering conference and symposium on the foundations of software engineering*. 2018, pp. 908–911.

[83] Matias Trapaglia et al. “Flexible Software to Hardware Migration Methodology for FPGA Design and Verification”. In: *2019 X Southern Conference on Programmable Logic (SPL)*. IEEE. 2019, pp. 39–44.

[84] Tampere University. *SoC-Hub*. 2025. URL: https://sochub.fi/.

[85] Janne Virtanen et al. “Kactus2: A graphical EDA tool built on the IP-XACT standard”. In: *The Journal of Open Source Software* 2.13 (2017), p. 1.

[86] Stefan Wagner. “Scrum for cyber-physical systems: a process proposal”. In: *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*. ACM. 2014, pp. 51–56.

[87] Edward Wang et al. “A methodology for reusable physical design”. In: *2020 21st International Symposium on Quality Electronic Design (ISQED)*. IEEE. 2020, pp. 243–249.

[88] Sheng-Hong Wang et al. “Livehd: A productive live hardware development flow”. In: *IEEE Micro* 40.4 (2020), pp. 67–75.

[89] Xi Wang et al. “ChatCPU: An Agile CPU Design & Verification Platform with LLM”. In: *61st ACM/IEEE Design Automation Conference (DAC’24)*. 2024, p. 6.

[90] Tianrui Wei et al. “Zoomie: A Software-like Debugging Tool for FPGAs”. In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 2024, pp. 1048–1062.

[91] Paul N Whatmough et al. “CHIPKIT: An agile, reusable open-source framework for rapid test chip development”. In: *IEEE Micro* 40.4 (2020), pp. 32–40.

[92] Binyi Wu et al. “Industry-track: Towards agile design of neural processing unit”. In: *2022 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE. 2022, pp. 17–20.

[93] Qingcheng Xiao et al. “Hasco: Towards agile hardware and software co-design for tensor computation”. In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2021, pp. 1055–1068.

[94] Lida Xu et al. “RIVL: A Low-Cost SoC Agile Development Platform for Multiple RISC-V Processors Design and Verification”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* (2024).

[95] Yinan Xu et al. “Towards developing high performance RISC-V processors using agile methodology”. In: *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE. 2022, pp. 1178–1199.

[96] Chen Yang et al. “Automated Python-to-RTL Transformation and Optimization for Neural Network Acceleration”. In: *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*. IEEE. 2024, pp. 253–258.

[97] Chunfeng Yuan et al. “Teaching undergraduates to build real computer systems”. In: *Communications of the ACM* 64.11 (2021), pp. 48–49.

[98] F. Zaruba and L. Benini. “The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.11 (Nov. 2019), pp. 2629–2640. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2019.2926114.

[99] Hongji Zou et al. “Towards implementing RTL microprocessor agile design using feature oriented programming”. In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2022, pp. 472–477.

[100] EHT Zurich and Università di Bologna. *PULP-Platform*. 2025. URL: https://pulp-platform.org/.