Leveraging Hardware Construction Languages for Flexible Design Space Exploration on FPGA

Bruno FERRES

Director: Frédéric ROUSSEAU, Prof. at Univ. Grenoble Alpes

Advisor: Olivier MULLER, Associate Prof. at Grenoble INP - Ensimag

System Level Synthesis - TIMA
Université Grenoble Alpes

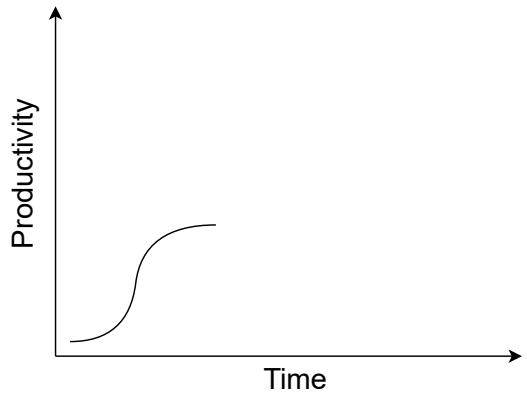
March 23, 2022

VCU128 board from Xilinx

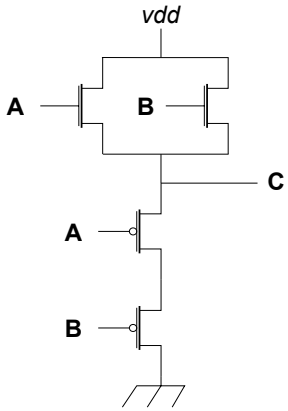
How to program this big circuit?

VCU128 board from Xilinx

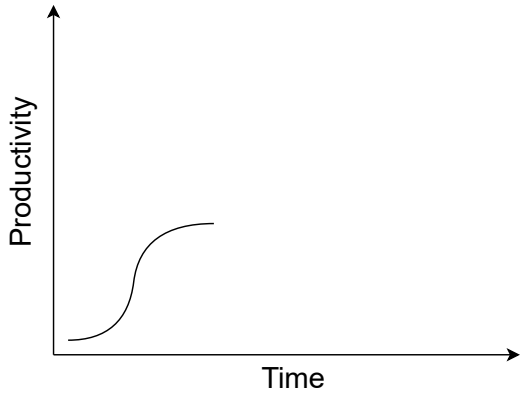
Design Methodologies and Productivity



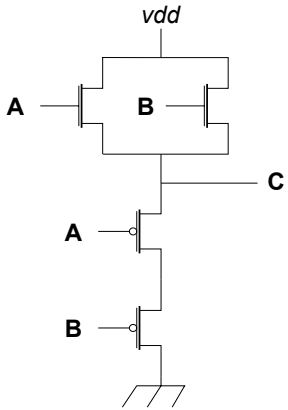
Design Methodologies and Productivity



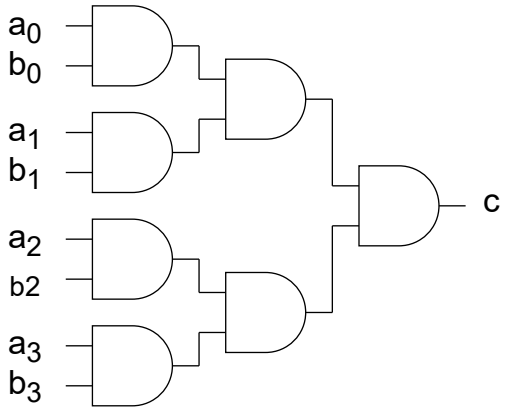
Circuit level



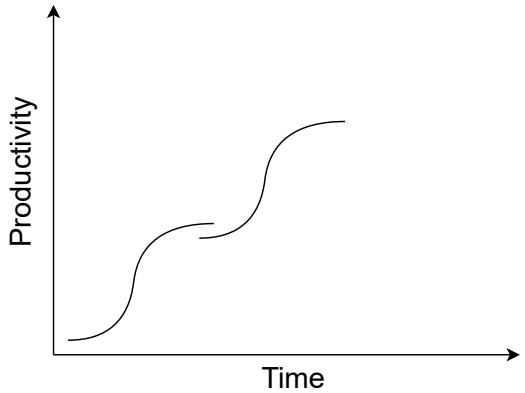
Design Methodologies and Productivity



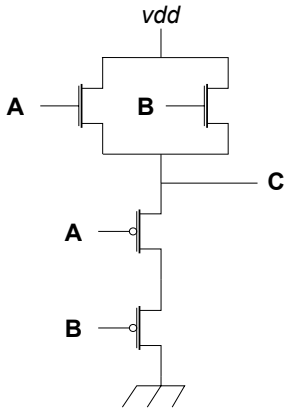
Circuit level



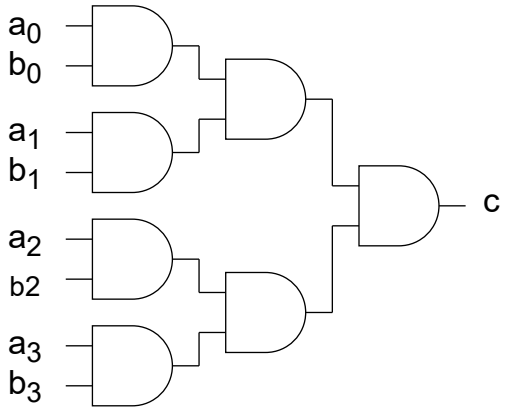
Logical level



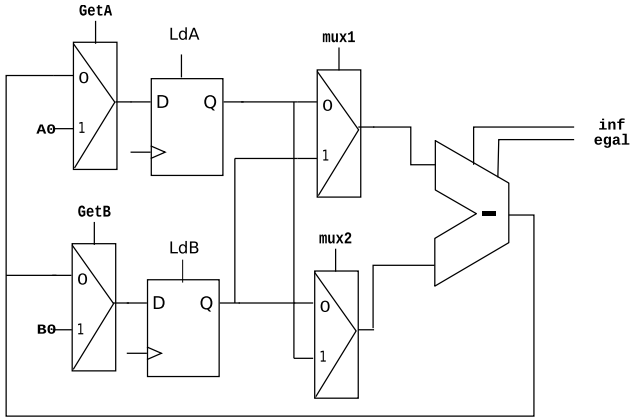
Design Methodologies and Productivity



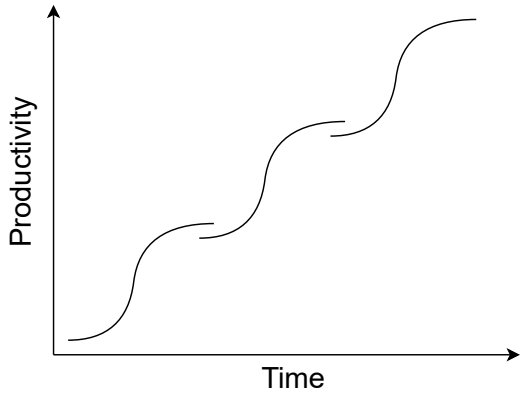
Circuit level



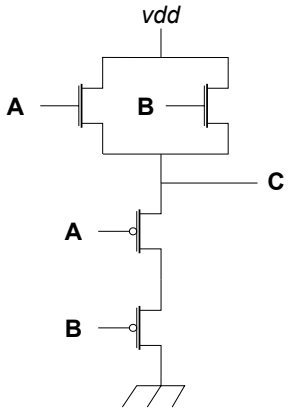
Logical level



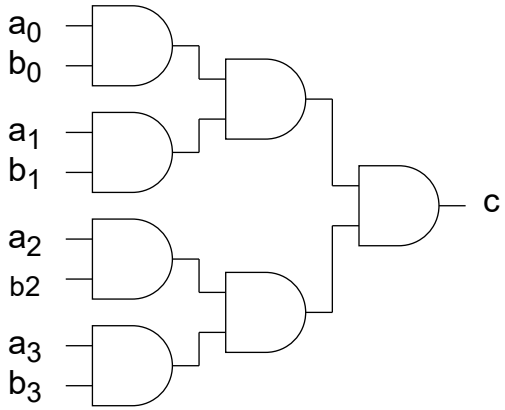
Register Transfer level



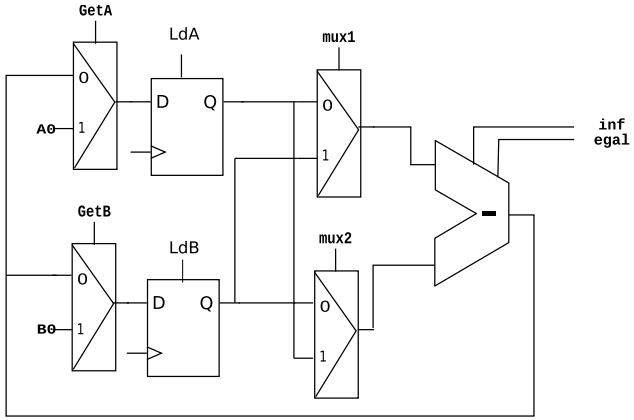
Design Methodologies and Productivity



Circuit level



Logical level

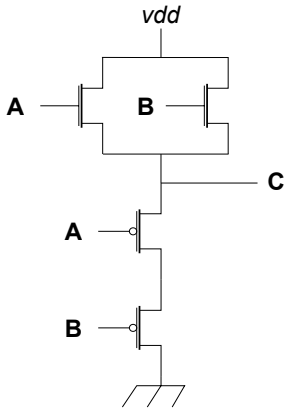


Register Transfer level

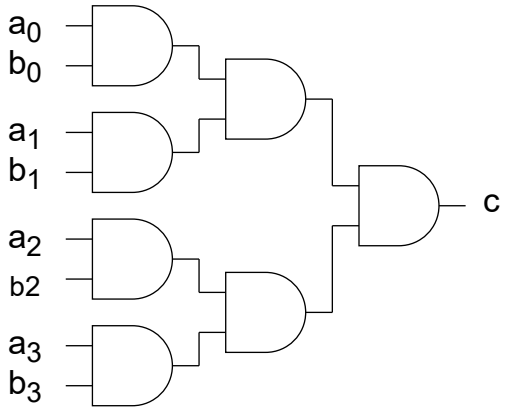
Improving hardware designers productivity

- which **innovations** do we rely on?

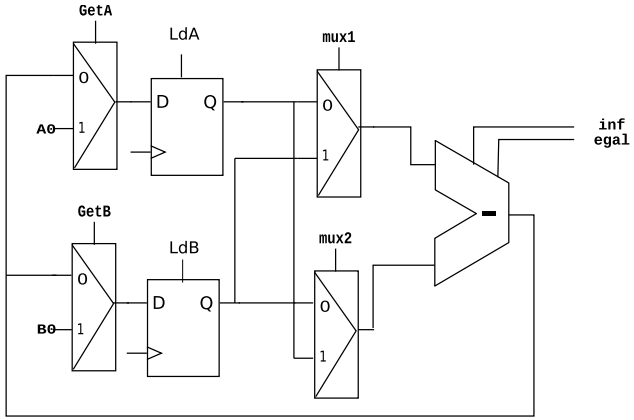
Design Methodologies and Productivity



Circuit level



Logical level



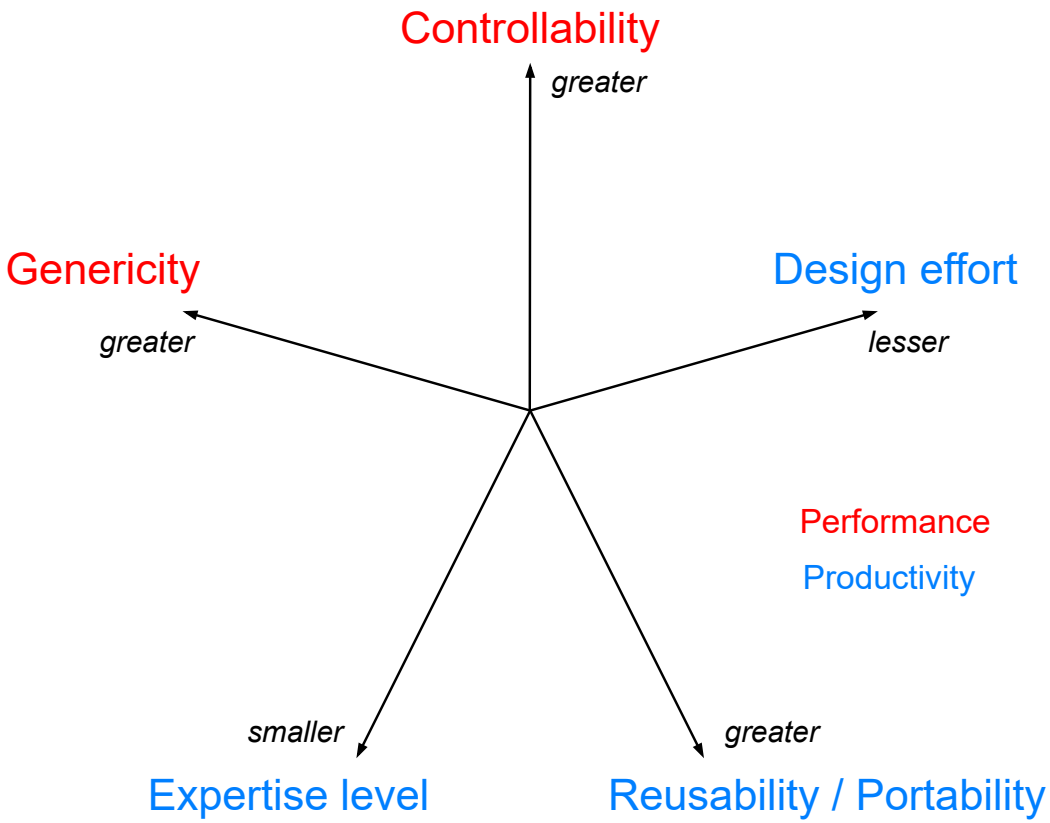
Register Transfer level

Improving hardware designers productivity

- which **innovations** do we rely on?
- what is **productivity** when it comes to **hardware design**?

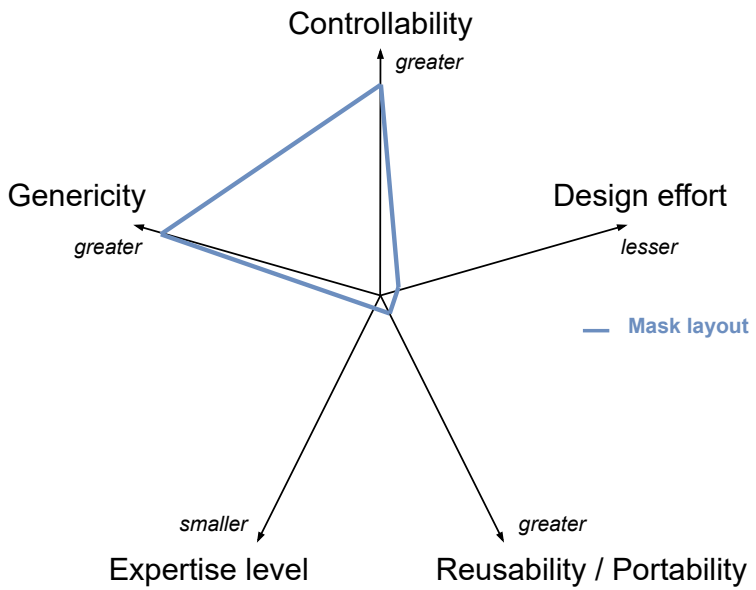
Plan

Characteristics of Design Methodologies



Productivity and **performance** criteria for hardware design

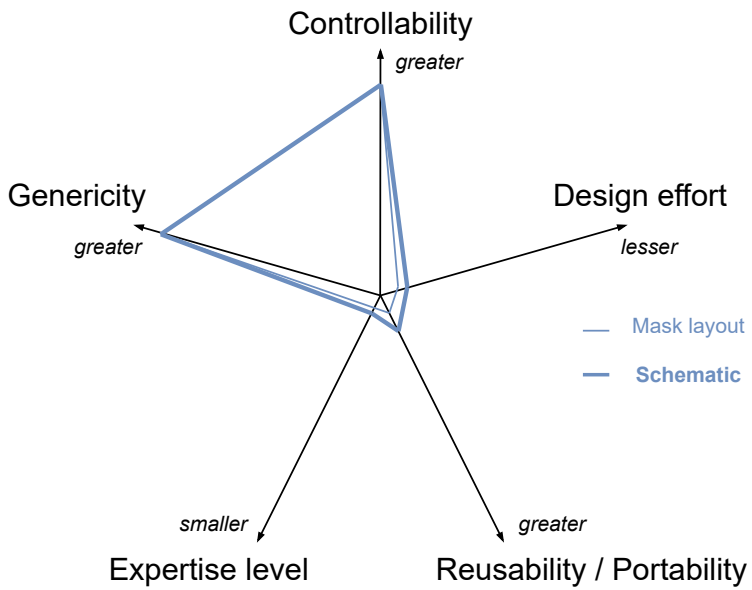
Digital Design Methodologies



A small history on digital design

- IC mask layout (circuit level)

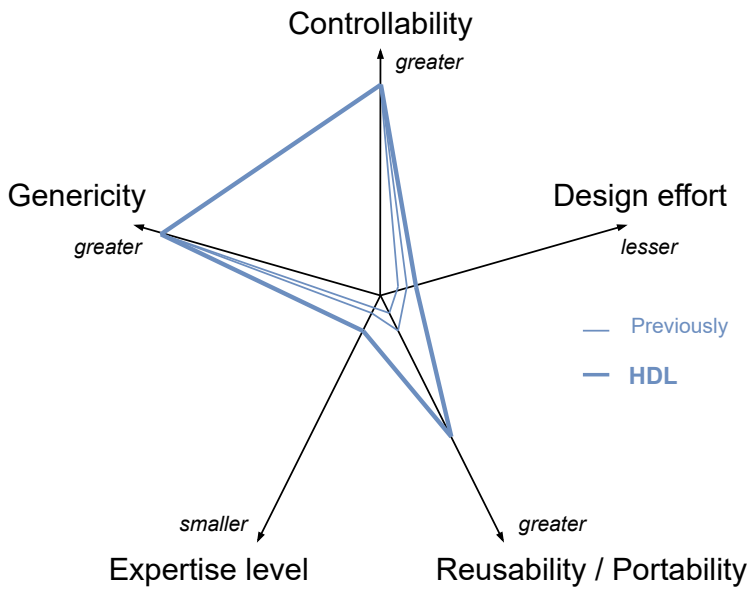
Digital Design Methodologies



A small history on digital design

- IC mask layout (circuit level)
- standard cells (logical level)

Digital Design Methodologies



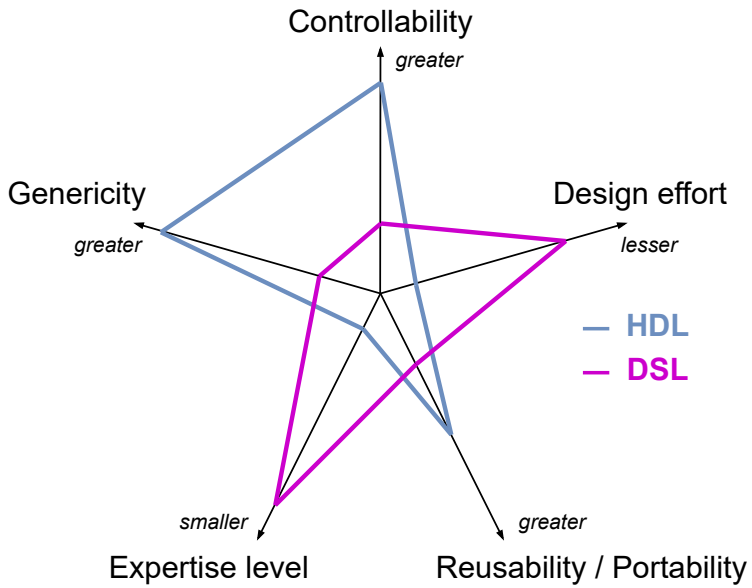
A small history on digital design

- IC mask layout (circuit level)
- standard cells (logical level)

Democratization of the HDLs

- **Hardware Description Languages** (Register Transfer Level)

Digital Design Methodologies



A small history on digital design

- IC mask layout (circuit level)
- standard cells (logical level)

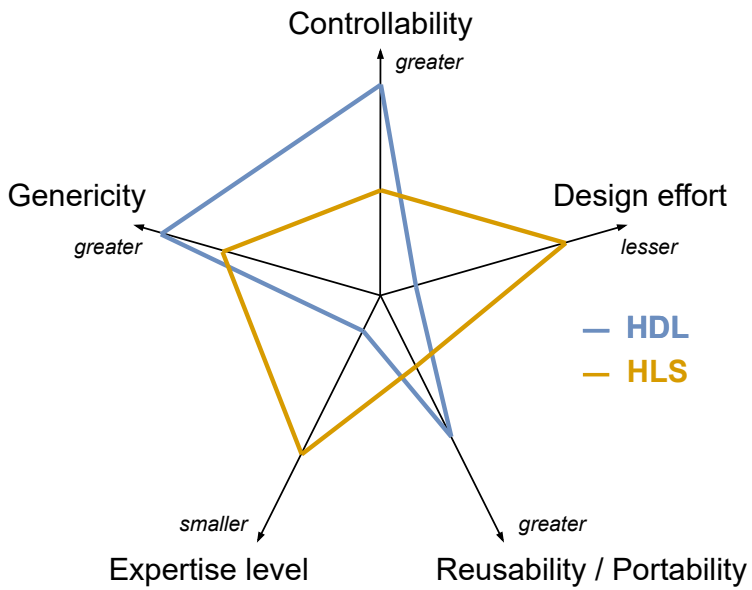
Democratization of the HDLs

- **Hardware Description Languages** (Register Transfer Level)

Existing alternatives (algorithmic level)

- **Domain Specific Languages (DSL)**

Digital Design Methodologies



A small history on digital design

- IC mask layout (circuit level)
- standard cells (logical level)

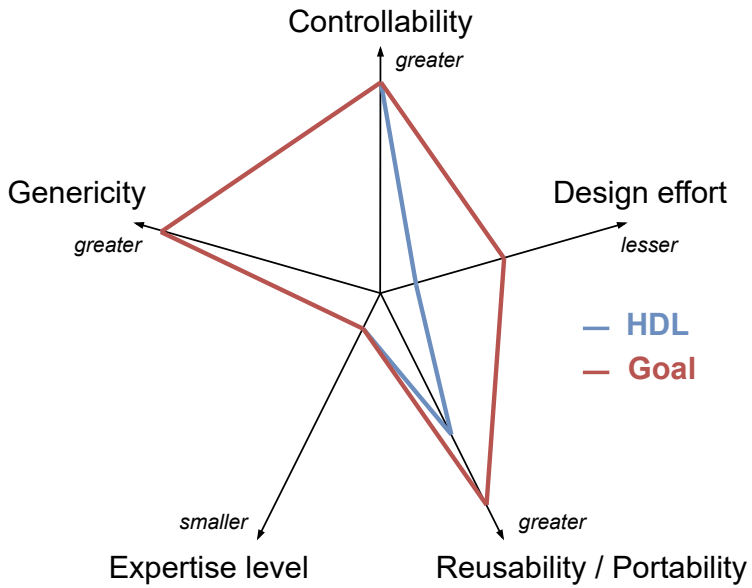
Democratization of the HDLs

- **Hardware Description Languages** (Register Transfer Level)

Existing alternatives (algorithmic level)

- **Domain Specific Languages (DSL)**
- **High Level Synthesis (HLS)**

Digital Design Methodologies



A small history on digital design

- IC mask layout (circuit level)
- standard cells (logical level)

Democratization of the HDLs

- **Hardware Description Languages** (Register Transfer Level)

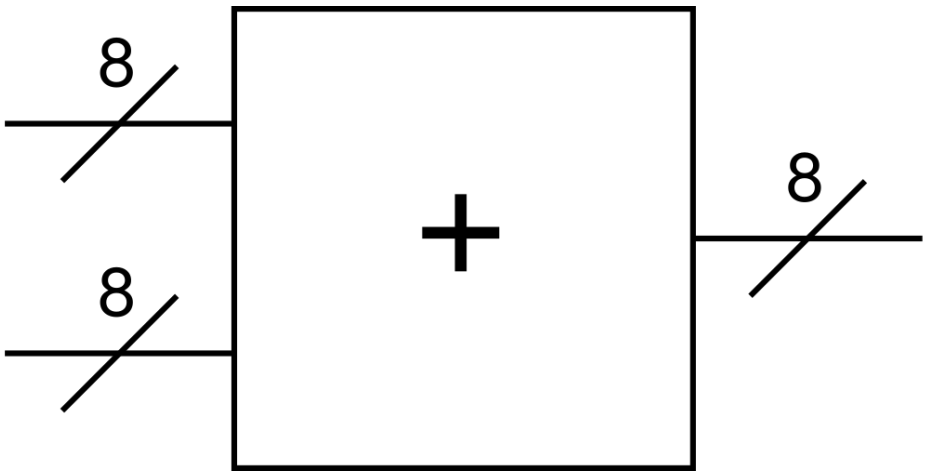
Existing alternatives (algorithmic level)

- **Domain Specific Languages (DSL)**
- **High Level Synthesis (HLS)**

What now?

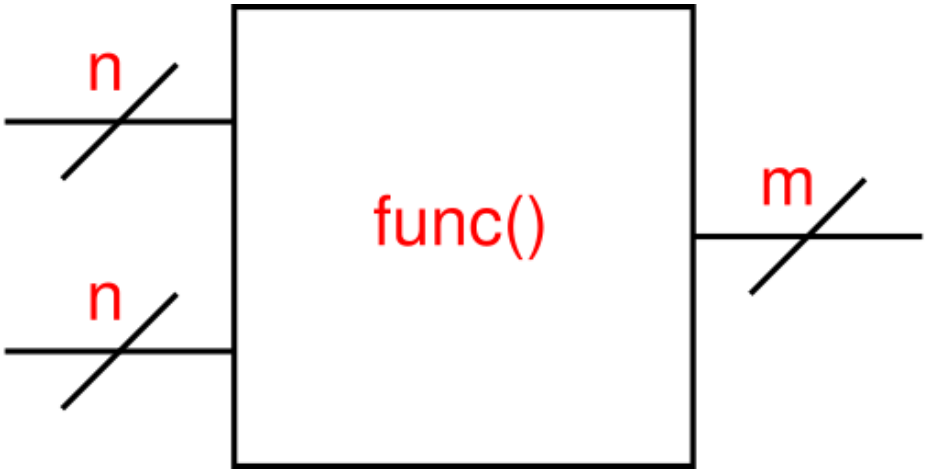
How to improve **productivity** while maintaining **performance**?

Improving Reusability through Hardware Construction



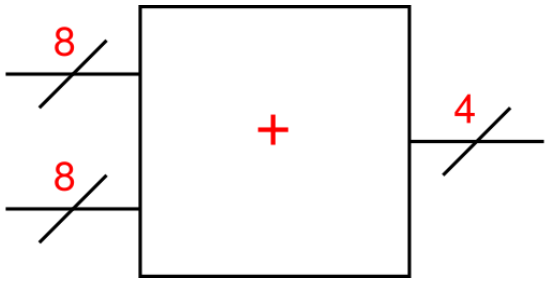
Simple 8 bit adder

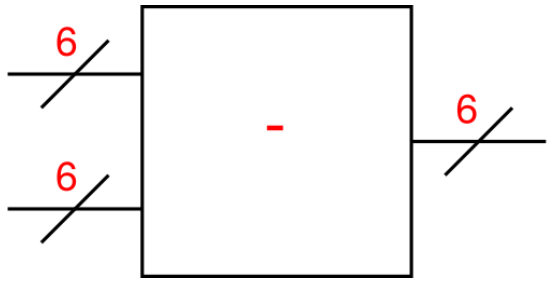
Improving Reusability through Hardware Construction



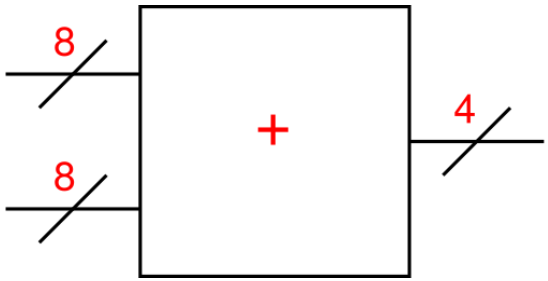
Parametrized functional block

Improving Reusability through Hardware Construction

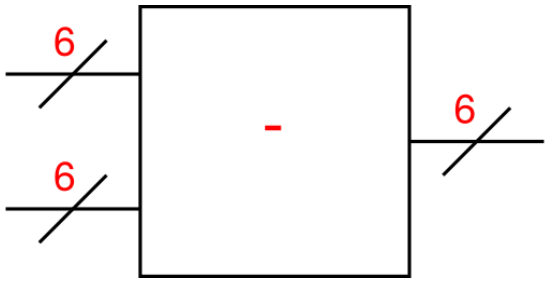




Improving Reusability through Hardware Construction

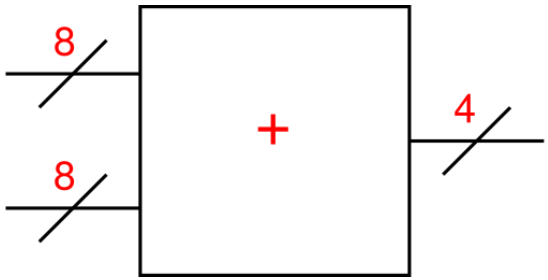



```
module adder (
    input [7:0] op1,
    input [7:0] op2,
    output [3:0] out
);
    wire [8:0] tmp;
    tmp = op1 + op2
    out = tmp > 15 ?
        15 : tmp[3:0] ;
end module
```

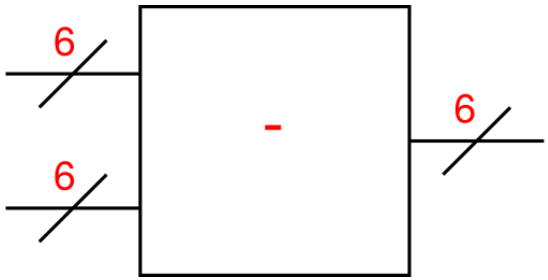



```
module sub (
    input [5:0] op1,
    input [5:0] op2,
    output [5:0] out
);
    out = op1 - op2;
end module
```

Improving Reusability through Hardware Construction




```
module adder (
    input [7:0] op1,
    input [7:0] op2,
    output [3:0] out
);
    wire [8:0] tmp;
    tmp = op1 + op2
    out = tmp > 15 ?
        15 : tmp[3:0] ;
end module
```

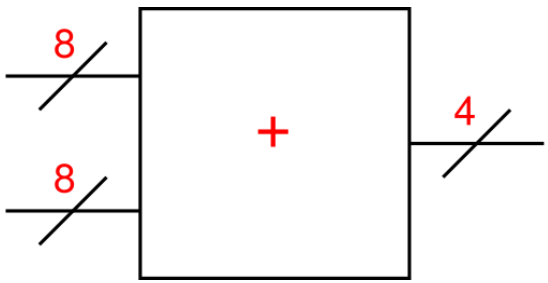


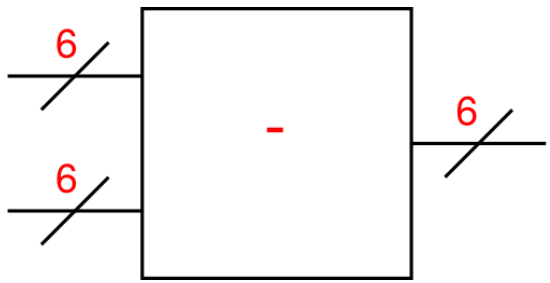

```
module sub (
    input [5:0] op1,
    input [5:0] op2,
    output [5:0] out
);
    out = op1 - op2;
end module
```

Hardware Description Languages features

Generic parameters could be used ... but not for everything!

Improving Reusability through Hardware Construction





Hardware Description limits

Same functionality, but limited sharing possible...

... **low reusability**!

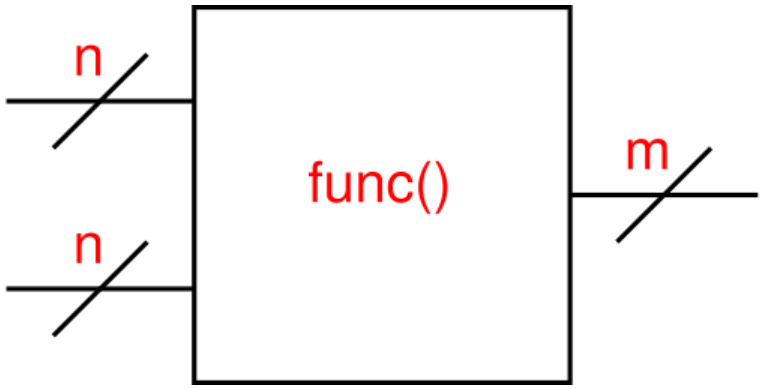
Interests of reusability

- **Reusable code = productivity increase**
- **Less code = less error = less debug!**

Improving Reusability through Hardware Construction

Hardware Construction Languages

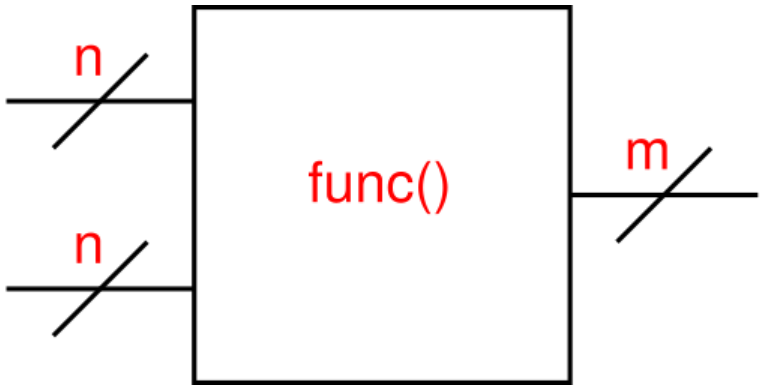
Designing **hardware generators** instead of *hardware accelerators*



Improving Reusability through Hardware Construction

Hardware Construction Languages

Designing **hardware generators** instead of *hardware accelerators*

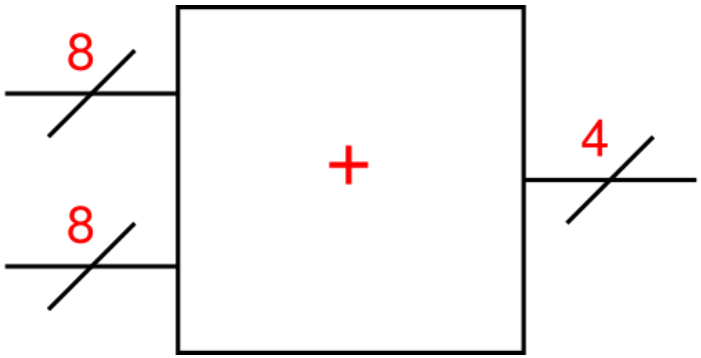


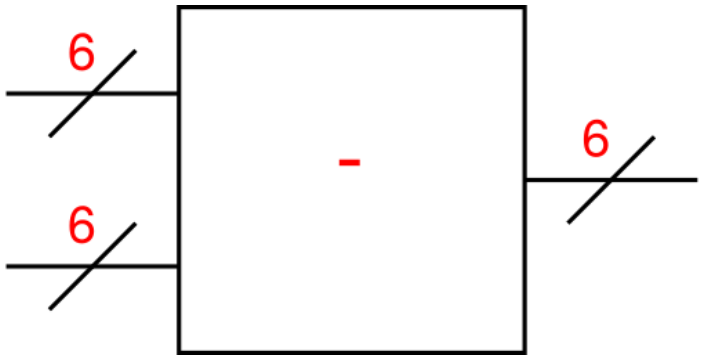

```
class CustomBlock[T <: Data](                    | Chisel syntax
    inputBitwidth:  Int,
    outputBitwidth: Int,
    func:           (T, T) => T
) extends Module { ... }
```

Improving Reusability through Hardware Construction

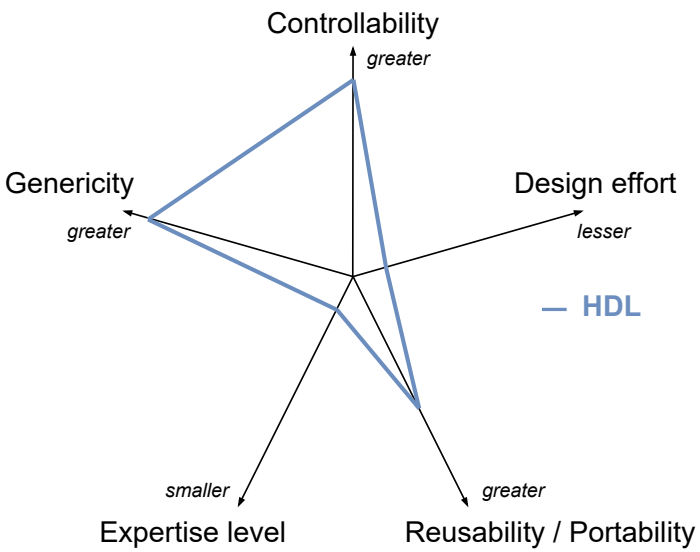
```
class CustomBlock[T <: Data](                          | Chisel syntax
    inputBitwidth:  Int,
    outputBitwidth: Int,
    func:           (T, T) => T
) extends Module { ... }

val adder    = new CustomBlock[UInt](8, 4, _ + _)
val sub      = new CustomBlock[UInt](6, 6, _ - _)
```



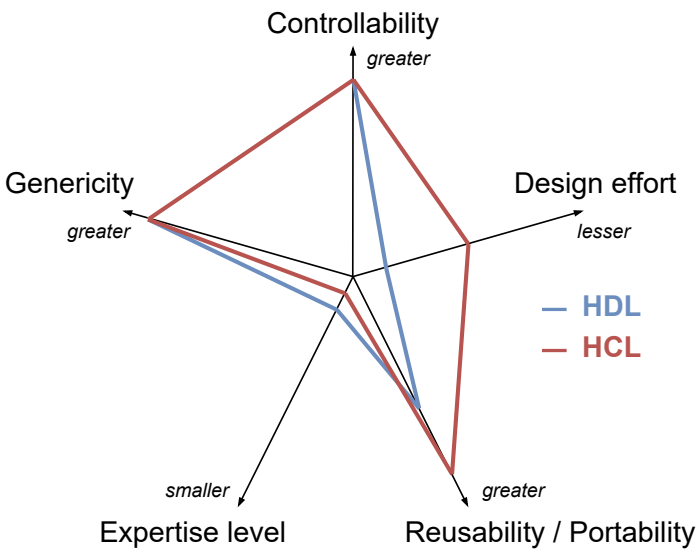


Hardware Construction Languages



HDL/HCL: Hardware Description / Construction Language

Hardware Construction Languages

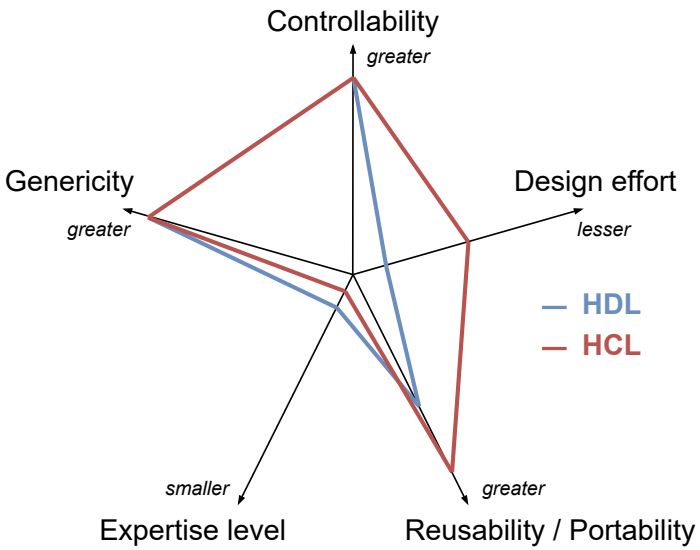


Hardware Construction Languages

- high reusability with **high level parameters**
- **controlled variations** of architectures
- **high level features**
  - *object oriented programming*

HDL/HCL: Hardware Description / Construction Language

Hardware Construction Languages



Hardware Construction Languages

- high reusability with **high level parameters**
- **controlled variations** of architectures
- **high level features**
  - *object oriented programming*

Chisel

- based on **Scala**
  - *functional programming*
- **open-source** framework
- **active community**!

HDL/HCL: Hardware Description / Construction Language

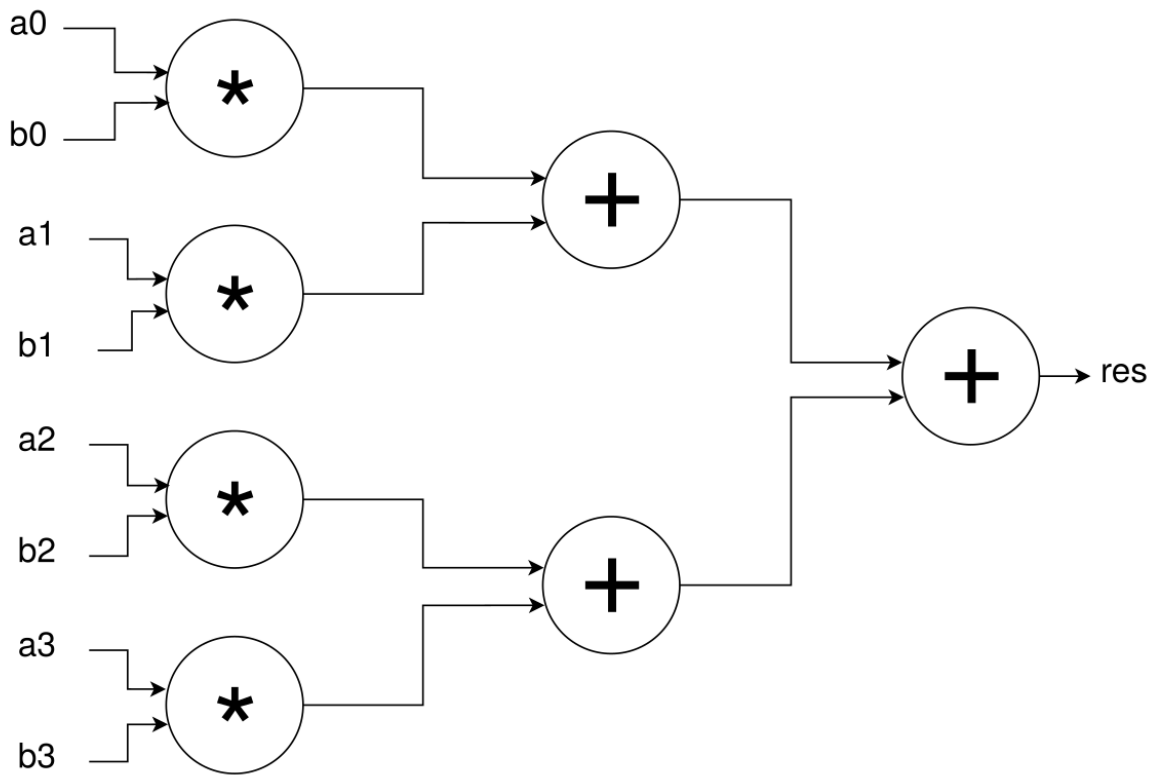
The Importance of Reusability: a Simple Usecase

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = a_0 b_0 + a_1 b_1 + a_2 b_2 + a_3 b_3$$

Dot Product computation

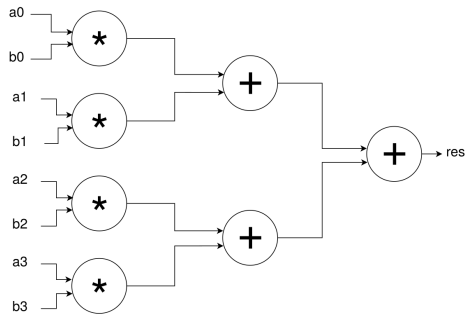
The Importance of Reusability: a Simple Usecase

$$a_0 b_0 + a_1 b_1 + a_2 b_2 + a_3 b_3$$



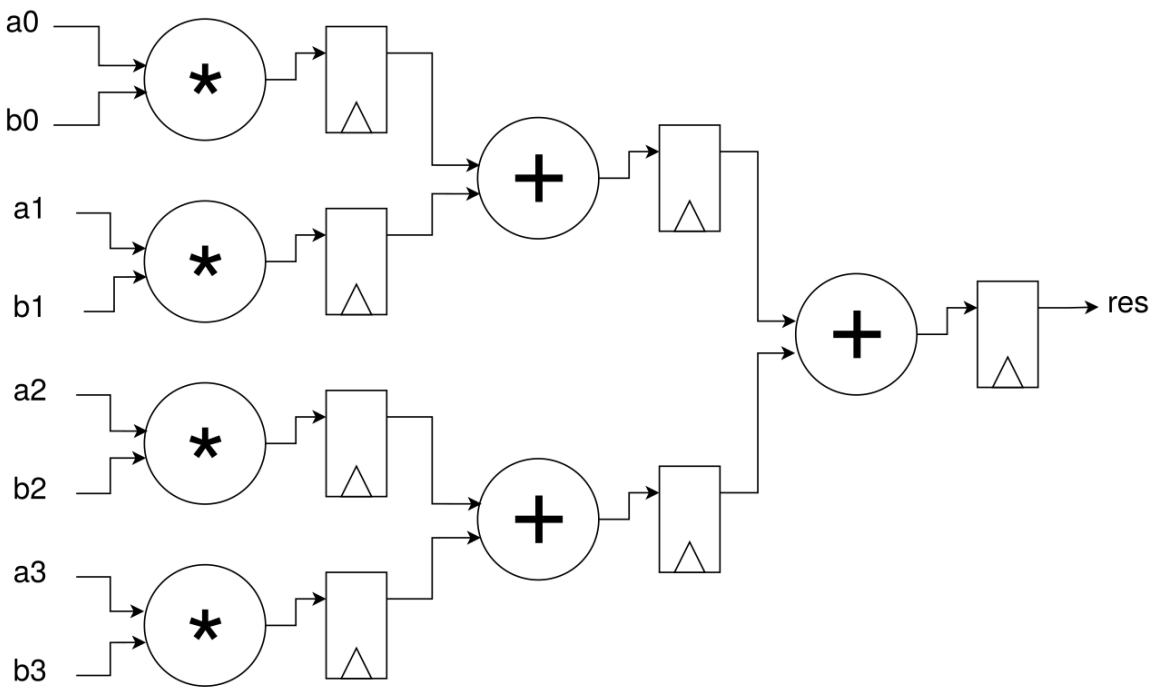
Architecture A

The Importance of Reusability: a Simple Usecase



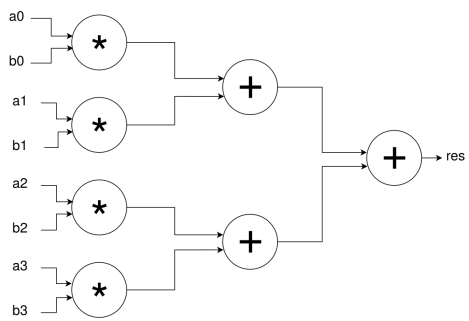
Architecture A

$$a_0 b_0 +$$
$$a_1 b_1 +$$
$$a_2 b_2 +$$
$$a_3 b_3$$

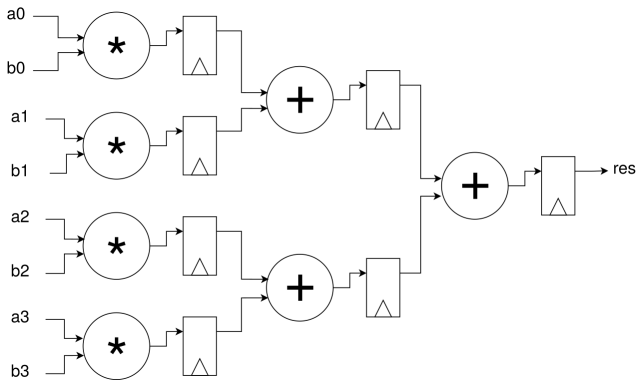


Architecture B

The Importance of Reusability: a Simple Usecase

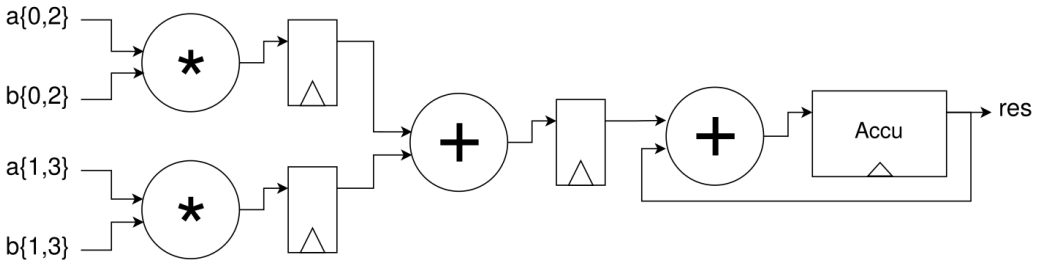


Architecture A



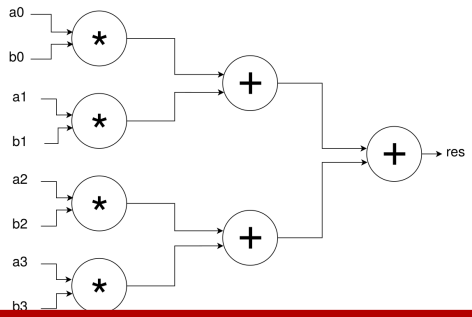
Architecture B

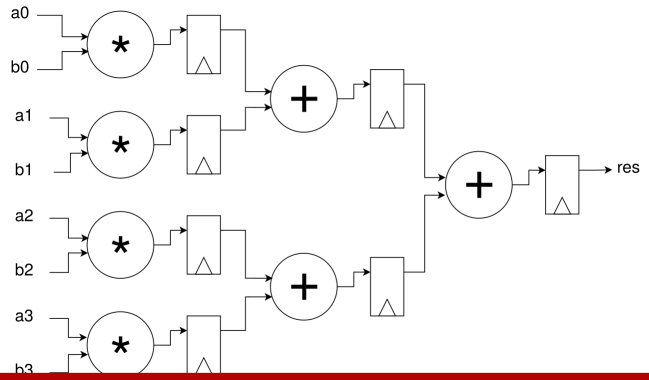
$$a_0 b_0 +$$
$$a_1 b_1 +$$
$$a_2 b_2 +$$
$$a_3 b_3$$



Architecture C

The Importance of Reusability: a Simple Usecase





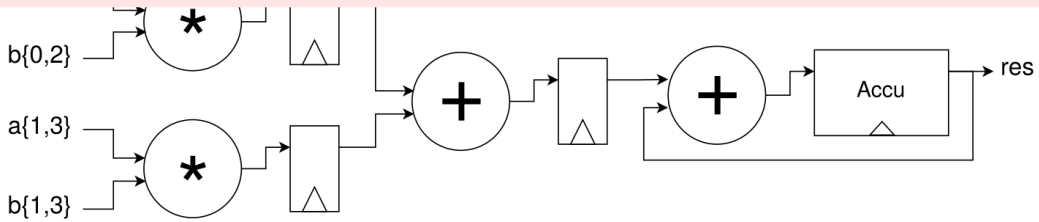
Problematic

- which **architecture** is the best for the usecase?
- how many **different descriptions** do we need?

$a_1 b_1 +$

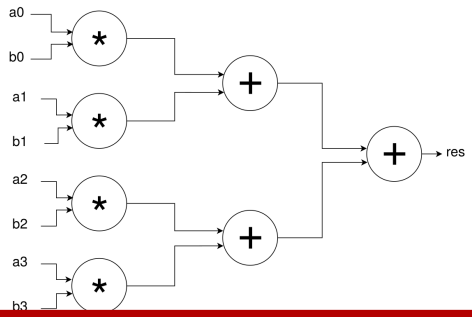
$a_2 b_2 +$

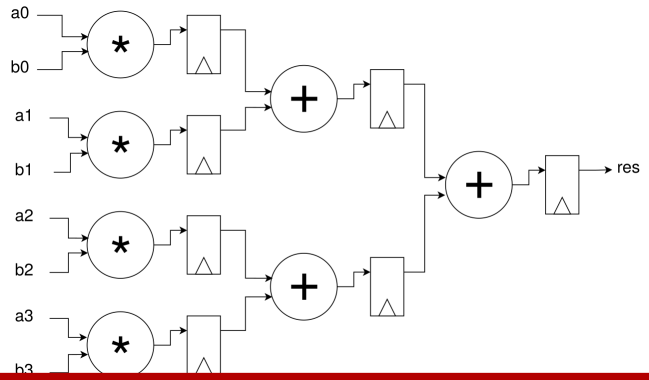
$a_3 b_3$



Architecture C

The Importance of Reusability: a Simple Usecase





Problematic

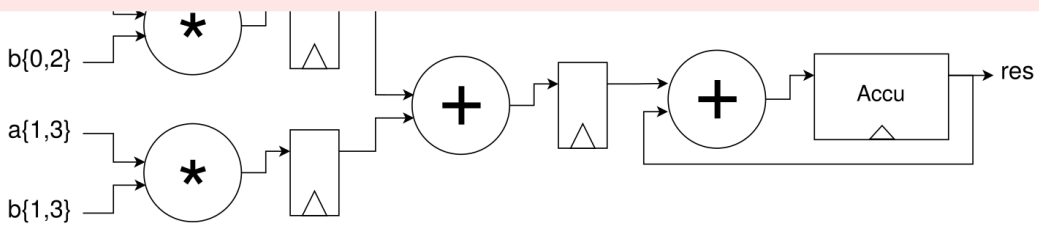
- which **architecture** is the best for the usecase?
- how many **different descriptions** do we need?

$\Rightarrow$ how to perform **Design Space Exploration**?

$a_1 b_1 +$

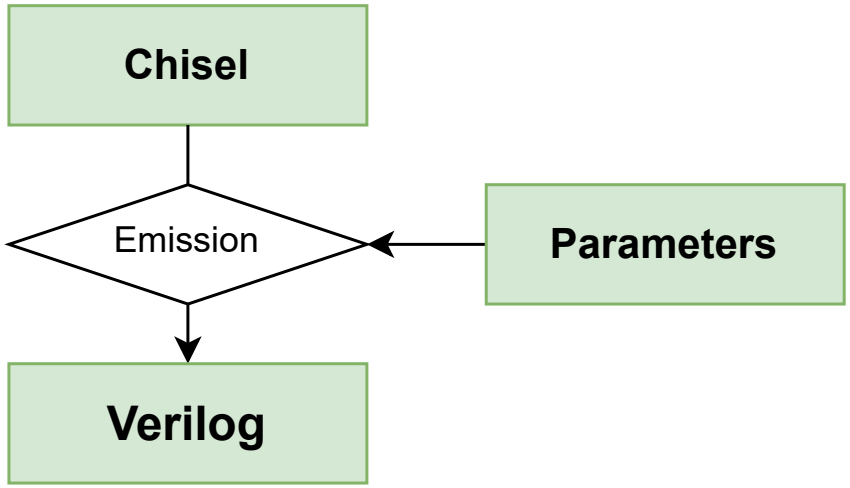
$a_2 b_2 +$

$a_3 b_3$



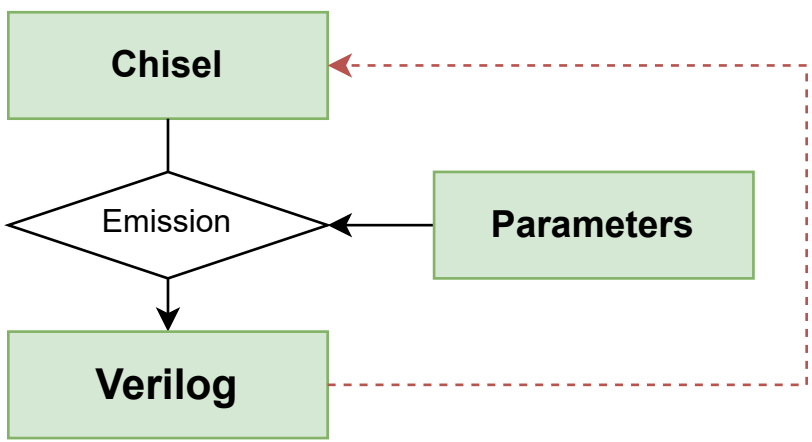
Architecture C

Hardware Construction Languages for Design Space Exploration



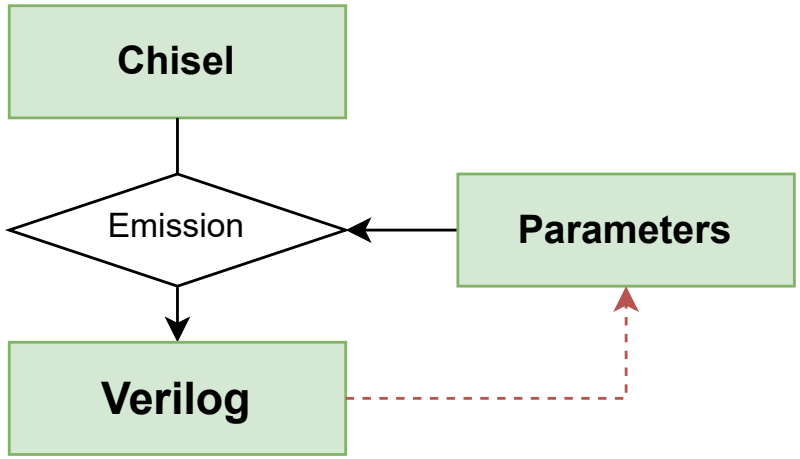
Standard Chisel emission flow

Hardware Construction Languages for Design Space Exploration



Iterating on the Chisel description

VS



Iterating on the emission parameters

Chisel for Design Space Exploration

Two **possible approaches**:

- iterate on the **Chisel description** itself
- exhibit **controlled variations of architecture** through **high level parameters**

Hardware Construction Languages for Design Space Exploration

Exploring the Dot Product Implementations

Identified parameters:

- use registers for pipelining?
- how many multipliers should be used in parallel?

Hardware Construction Languages for Design Space Exploration

Exploring the Dot Product Implementations

Identified parameters:

- use registers for pipelining?
- how many multipliers should be used in parallel?

```
class DotProduct(                                  | Chisel syntax
    useRegisters:   Boolean,
    parallelism:    Int
) extends Module { ... }
```

Motivations

Problem Statement

- What can HCLs bring to hardware developers?
- How can HCLs help building efficient DSE processes?

Motivations

Problem Statement

- What can HCLs bring to hardware developers?
- How can HCLs help building efficient DSE processes?

Positioning

- **Public:** **hardware developers / design experts**
- **Target:** *Field-Programmable Gate Array* devices
- **Goal:** increase **reusability** of designs and provide features to build **expertise-based** design processes
- **Approach:** provide a **flexible design space exploration framework** based on the usage of **Chisel**

Plan

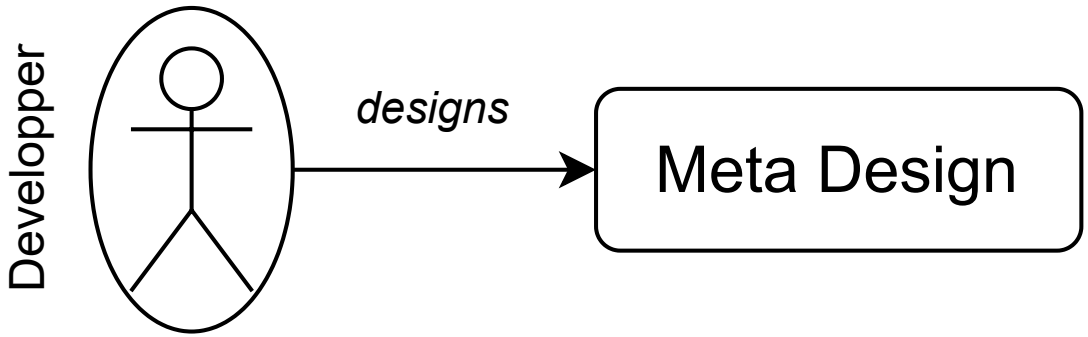
Contributions

Dual-methodology for efficient **HCL-based DSE**:

Contributions

Dual-methodology for efficient **HCL-based DSE**:

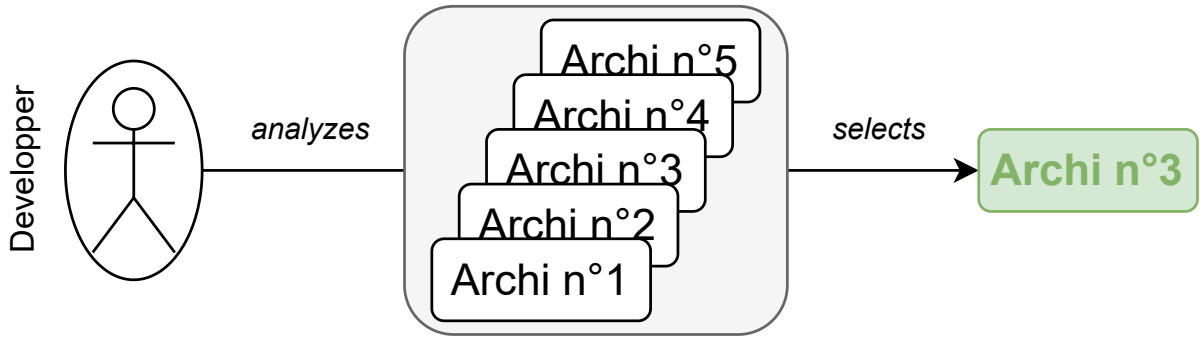
- **meta design** — how to build explorable generators?
  - how to define the useful parameters?
  - how to use them to build a generator?



Contributions

Dual-methodology for efficient **HCL-based DSE**:

- **meta design** — how to build explorable generators?
  - how to define the useful parameters?
  - how to use them to build a generator?
- **meta exploration** — how to efficiently explore them?
  - how to expose the design space?
  - how to estimate the quality of an implementation?
  - how to scan and compare multiple implementations?



Meta Design

Generator design process based on **prior analysis** of algorithm and target.

Meta Design

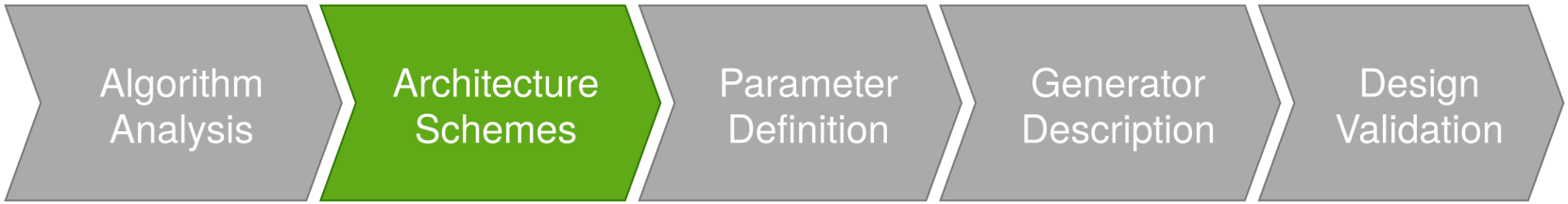
Generator design process based on **prior analysis** of algorithm and target.

Algorithm Analysis

Study the implemented algorithm, and its interactions with the target.

Meta Design

Generator design process based on **prior analysis** of algorithm and target.



Architecture Schemes

Define the **hierarchy of components** to be used in the built generator.

Meta Design

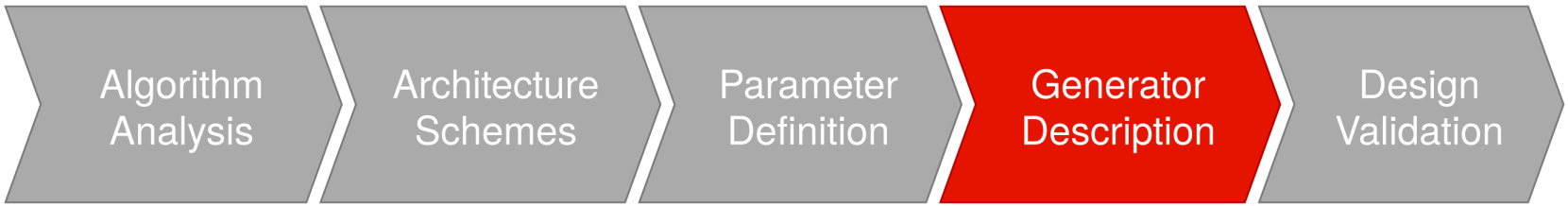
Generator design process based on **prior analysis** of algorithm and target.

Parameter Definition

Exhibit **high level parameters** to build meaningful generators.

Meta Design

Generator design process based on **prior analysis** of algorithm and target.

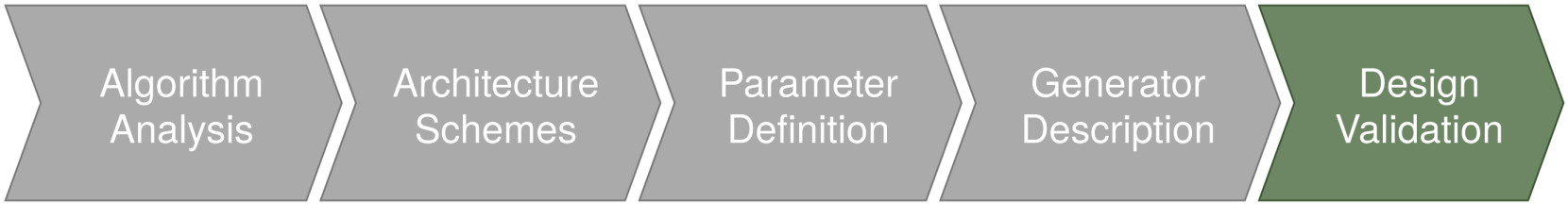


Generator Description

Describe an **accelerator generator** based on the defined parameters.

Meta Design

Generator design process based on **prior analysis** of algorithm and target.



Design Validation

Validate the behaviour of (some of) the generated accelerators.

Simple Usecase

Algorithm: GEMM (General Matrix Multiply)

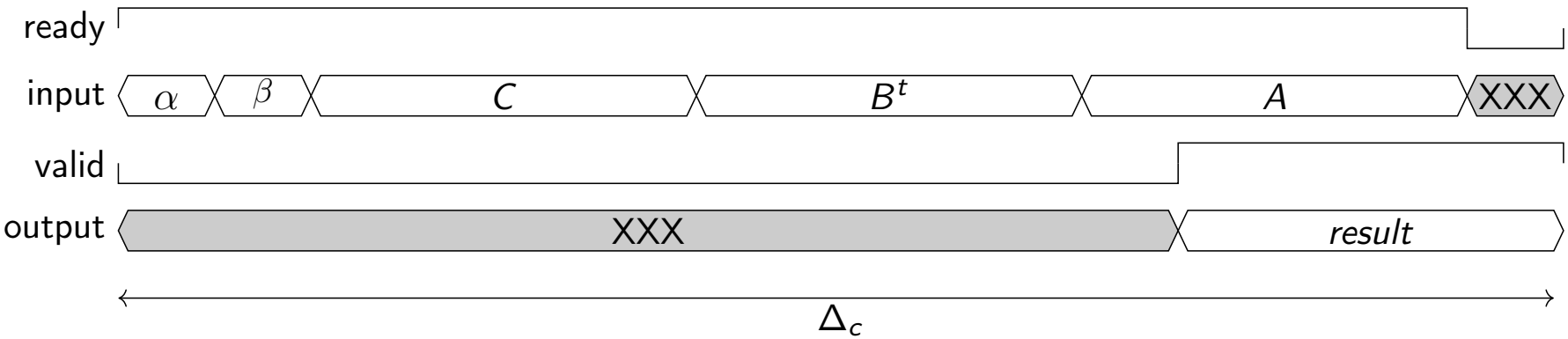
$$f : \mathbb{N} \times \mathbb{N} \times \mathcal{M}_n \times \mathcal{M}_n \times \mathcal{M}_n \to \mathcal{M}_n$$
$$(\alpha, \beta, A, B, C) \mapsto \alpha \cdot A \times B + \beta \cdot C$$

Simple Usecase

Algorithm: GEMM

$$\alpha \cdot A \times B + \beta \cdot C$$

Targeted temporal behaviour:



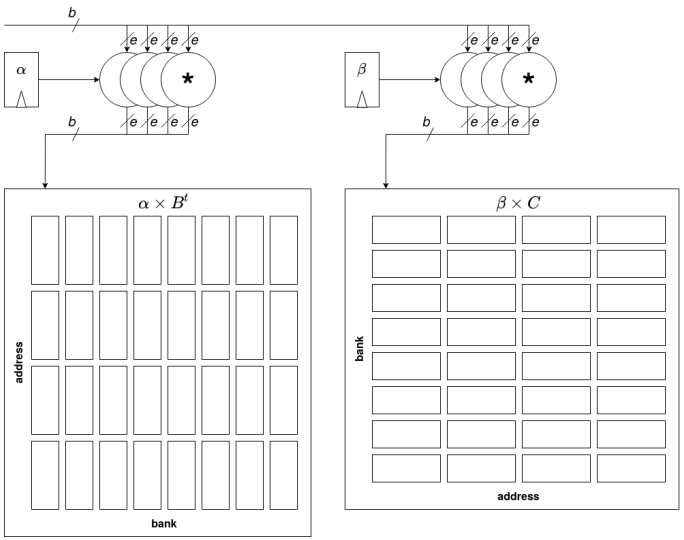
Using **users expertise** for **algorithm analysis**

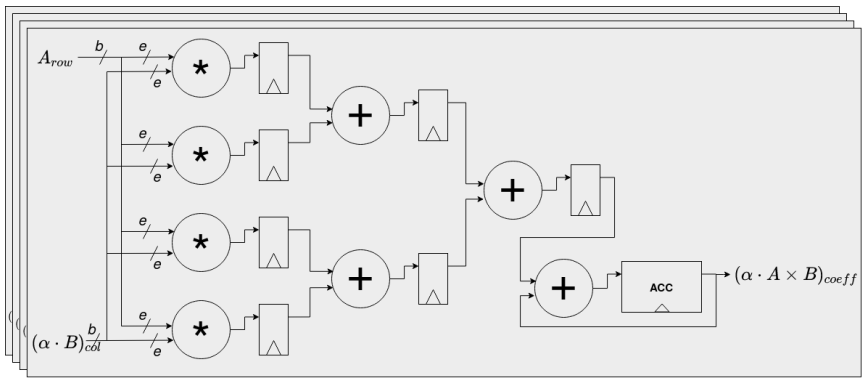
Simple Usecase

Algorithm: GEMM

$$\alpha \cdot A \times B + \beta \cdot C$$

Architecture choices:





xn

Simple Usecase

Three parameters identified:

- **architectural** parameters
  - capacity of the bus
  - width of the elements
- **applicative** parameters
  - matrix dimension

Simple Usecase

Three parameters identified:

- **architectural** parameters
  - capacity of the bus
  - width of the elements
- **applicative** parameters
  - matrix dimension


```
class GemmModule(
    busWidth:       Int,
    elementWidth:   Int,
    dimension:      Int
) extends Module {...}
```

Simple Usecase

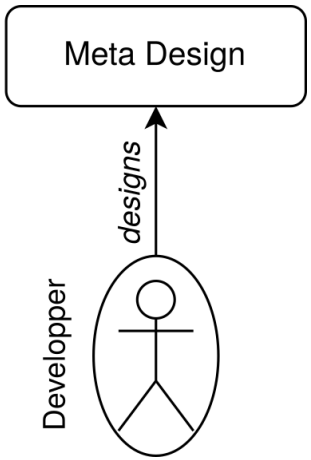
Three parameters identified:

- **architectural** parameters
  - capacity of the bus
  - width of the elements
- **applicative** parameters
  - matrix dimension


```
class GemmModule(
    busWidth:       Int,
    elementWidth:   Int,
    dimension:      Int
) extends Module {...}
```

Design validation is not much changed with respect to HDL flows

Meta Exploration

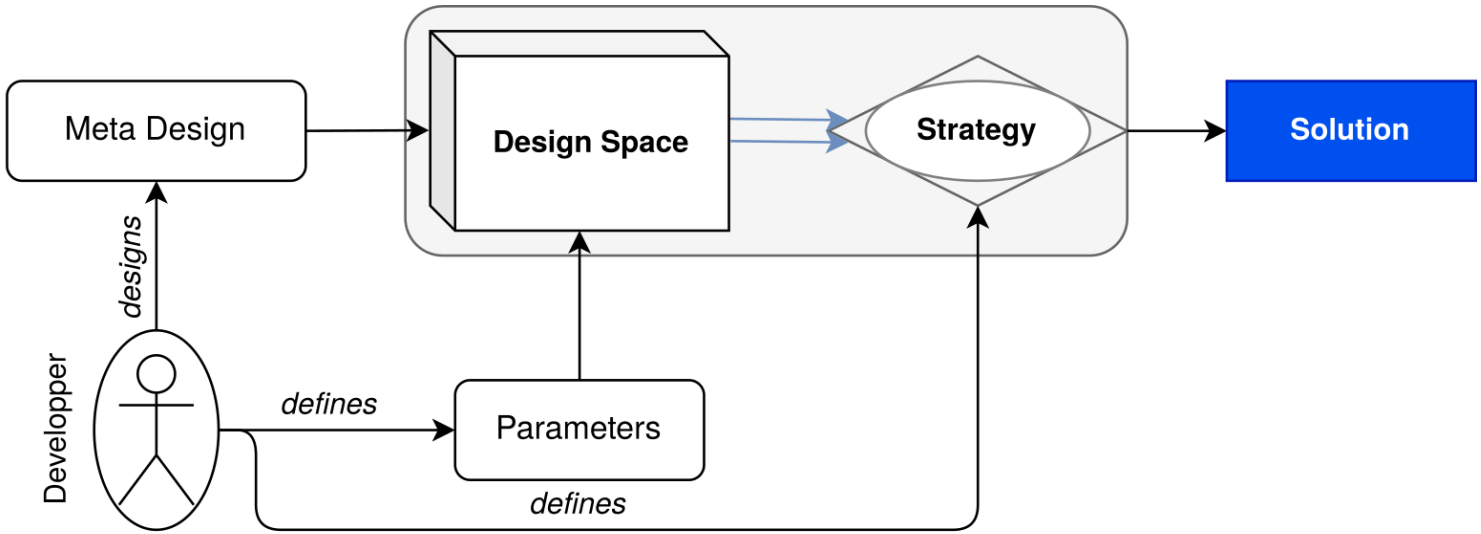


From Meta Design to Meta Exploration

How to efficiently use the built generators for **exploration**?

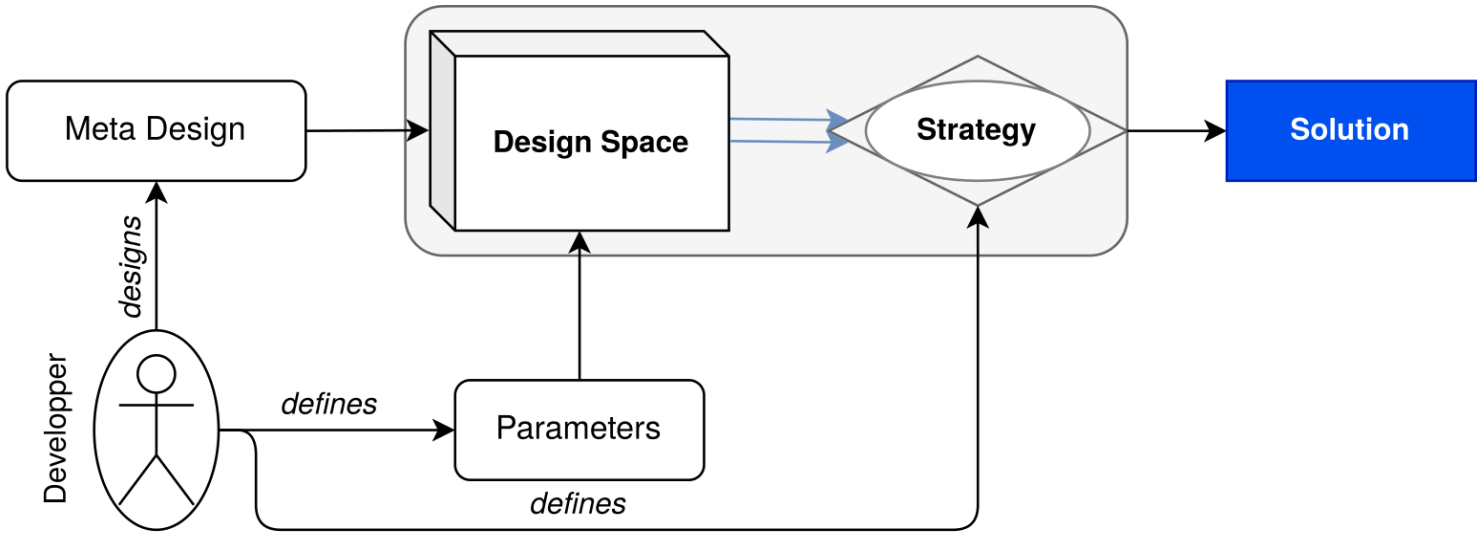
Meta Exploration

Exploration process exploiting both **exposed design space** and **user knowledge**.



Meta Exploration

Exploration process exploiting both **exposed design space** and **user knowledge**.



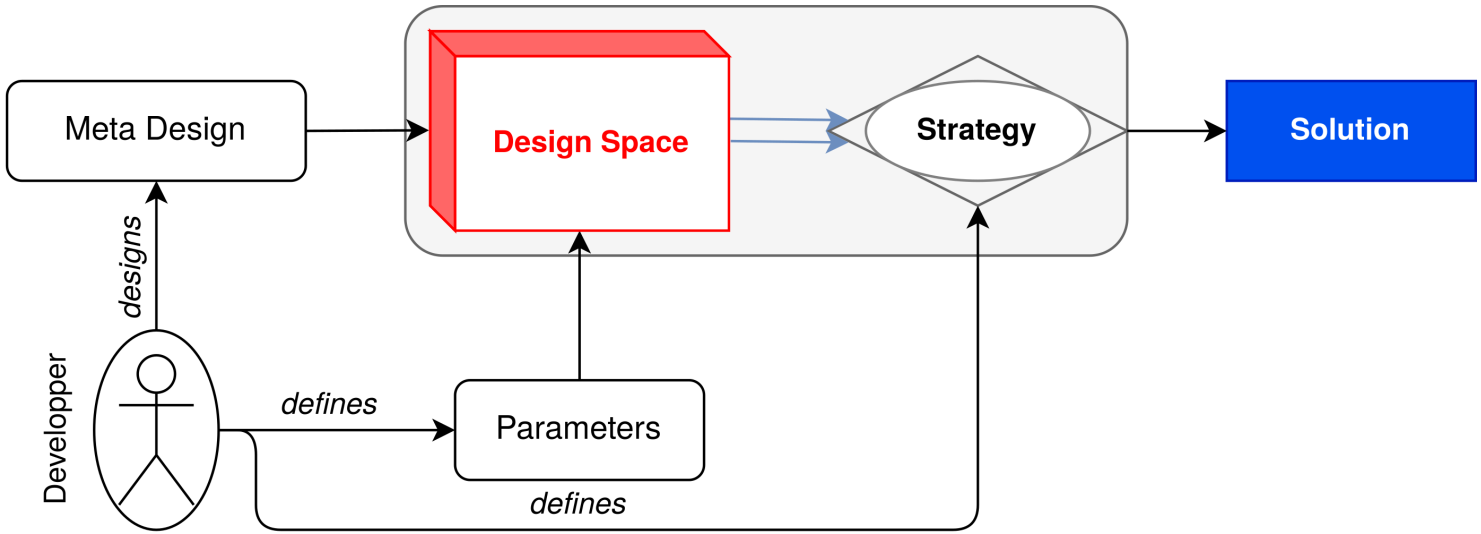
Two step approach

- expose the design space to explore
- define how to explore it for a particular use case

Meta Exploration — Design Space Exposition

Designer-based approach

Expose an **interesting design space** to be explored.



Simple Usecase

Exposing the meta design space:

```
class GemmModule(
    @pow2(5, 10) busWidth:      Int,
    @enum(32)    elementWidth:  Int,
    @pow2(4, 10) dimension:     Int
) extends Module with Explorable
```

Simple Usecase

Exposing the meta design space:

```
class GemmModule(
    @pow2(5, 10) busWidth:      Int,
    @enum(32)    elementWidth:  Int,
    @pow2(4, 10) dimension:     Int
) extends Module with Explorable
```

- `busWidth` $\in \{2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}\}$ (6 values)
- `elementWidth` $\in \{32\}$ (1 value)
- `dimension` $\in \{2^4, 2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}\}$ (7 values)

Simple Usecase

Exposing the meta design space:

```
class GemmModule(
    @pow2(5, 10) busWidth:      Int,
    @enum(32)    elementWidth:  Int,
    @pow2(4, 10) dimension:     Int
) extends Module with Explorable
```

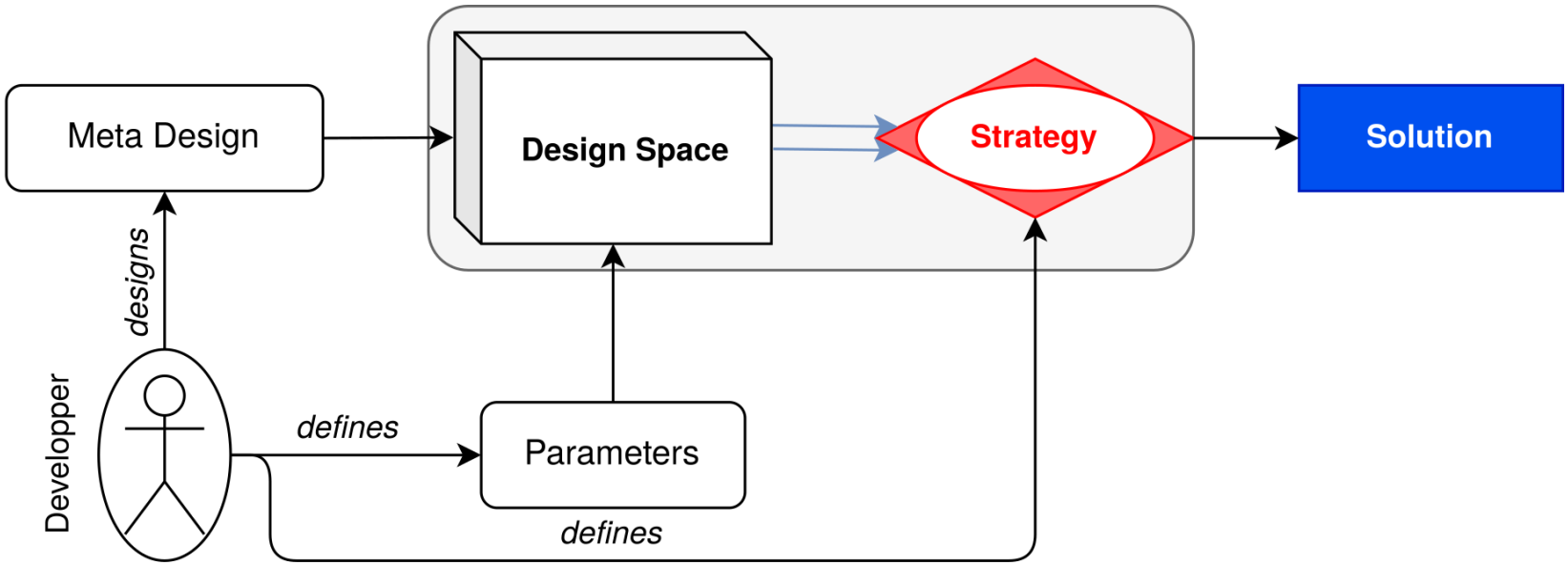
- `busWidth` $\in \{2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}\}$ (6 values)
- `elementWidth` $\in \{32\}$ (1 value)
- `dimension` $\in \{2^4, 2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}\}$ (7 values)

A $6 \times 7 = 42$ **implementations wide** design space is exposed

Meta Exploration — Strategy Definition

Functional approach

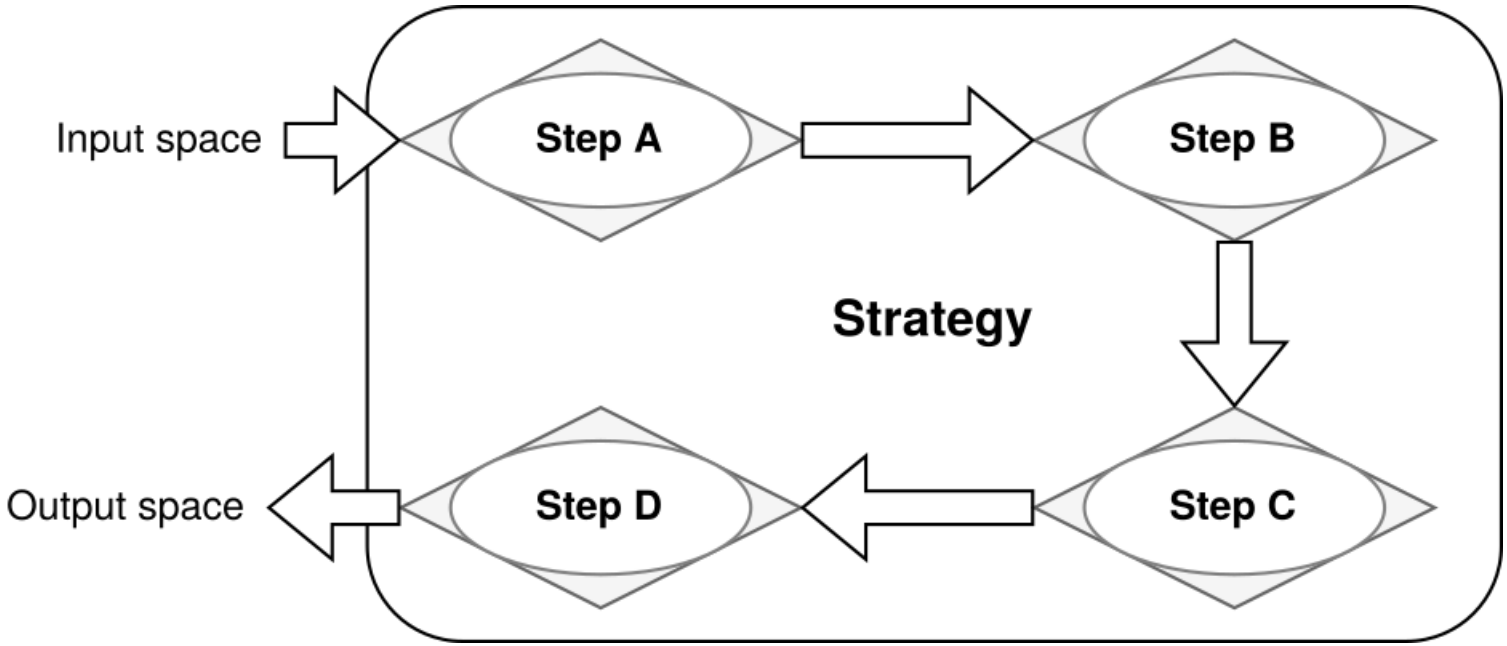
Describe an **exploration strategy** as a composition of **basic steps**



Meta Exploration — Strategy Definition

Functional approach

Describe an **exploration strategy** as a composition of **basic steps**



Meta Exploration — Strategy Definition

Functional approach

Describe an **exploration strategy** as a composition of **basic steps**

Each step relies on **two main notions**:

- the **metrics of interest** and their **estimators**
- the algorithm to **scan** the design space and produce a new one

Simple Usecase

Algorithm analysis:

Matrix multiply algorithm is computation intensive.

Simple Usecase

Algorithm analysis:

Matrix multiply algorithm is computation intensive.

Designer expertise

- **FPGA target & limited memory model**

$\Rightarrow$

Impact on the Strategy

- usage of LUT/DSP over FF/BRAM

Simple Usecase

Algorithm analysis:

Matrix multiply algorithm is computation intensive.

Designer expertise

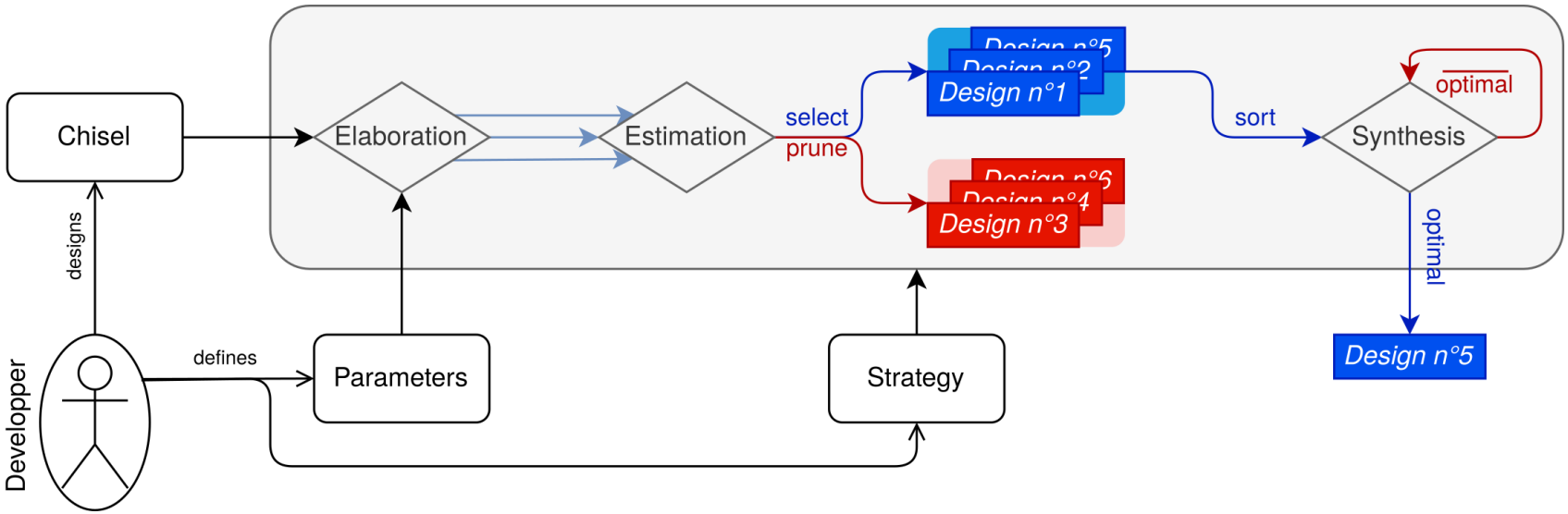
- **FPGA target** & **limited memory model**
- **32 bits** elements

$\Rightarrow$

Impact on the Strategy

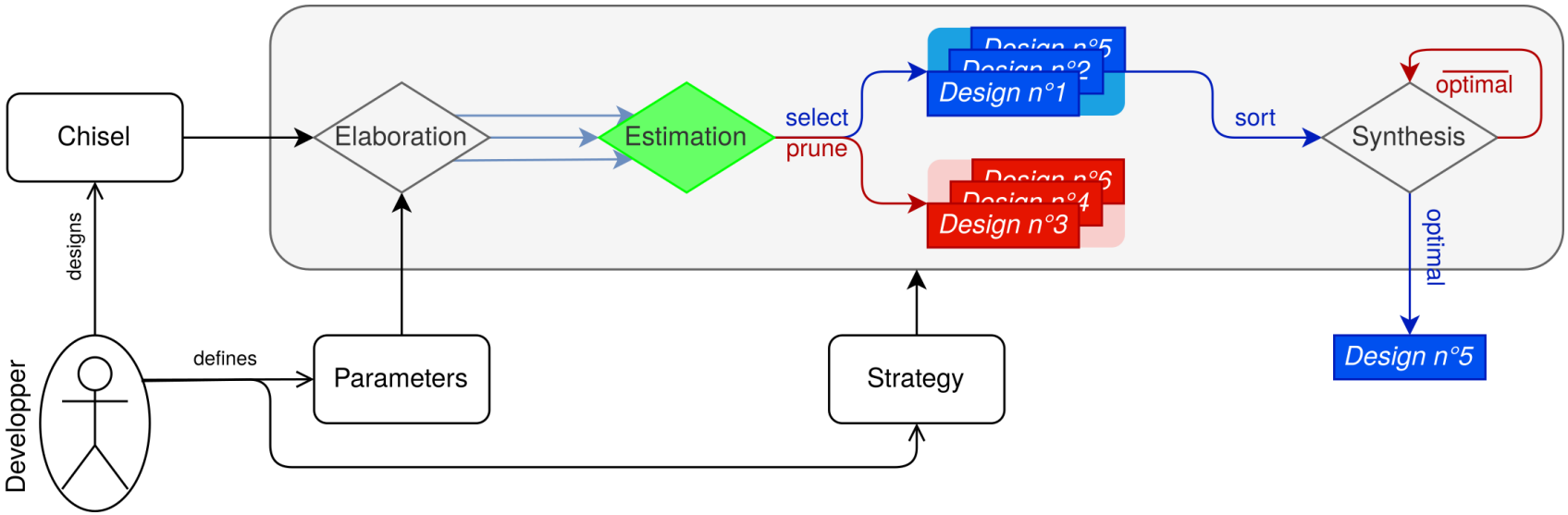
- usage of LUT/DSP over FF/BRAM
- usage of DSP over LUT

Simple Usecase



Example of meta exploration strategy

Simple Usecase

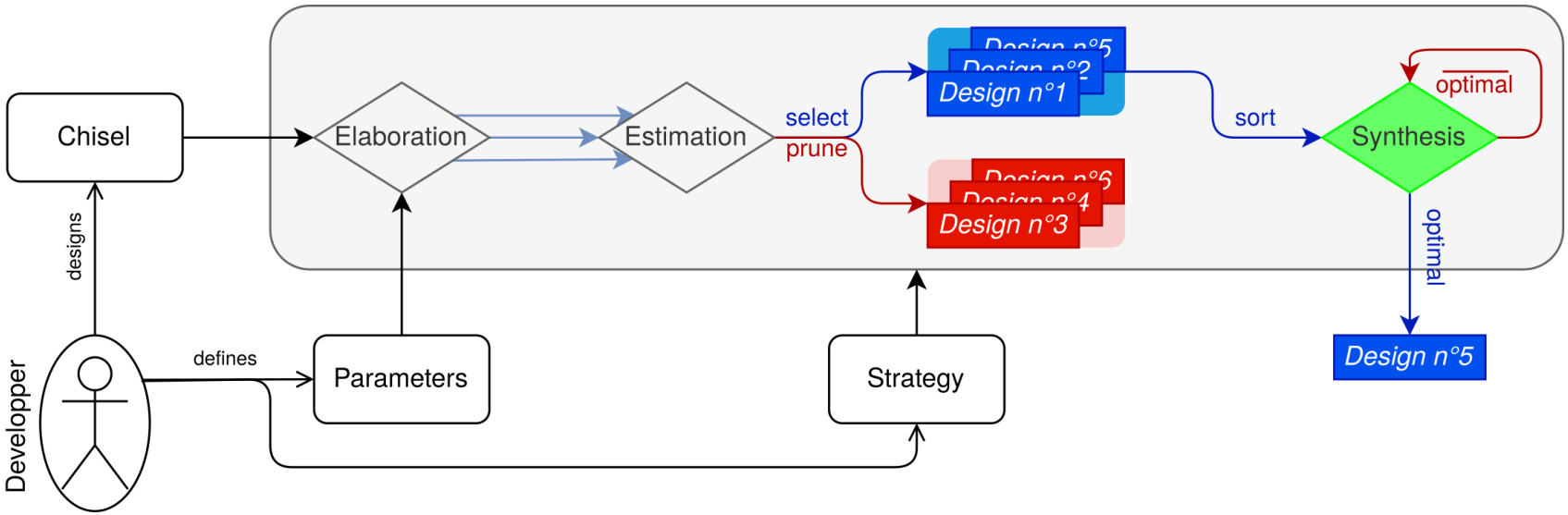


Example of meta exploration strategy

Step 1

Estimate the DSP usage at high level for efficient pruning

Simple Usecase

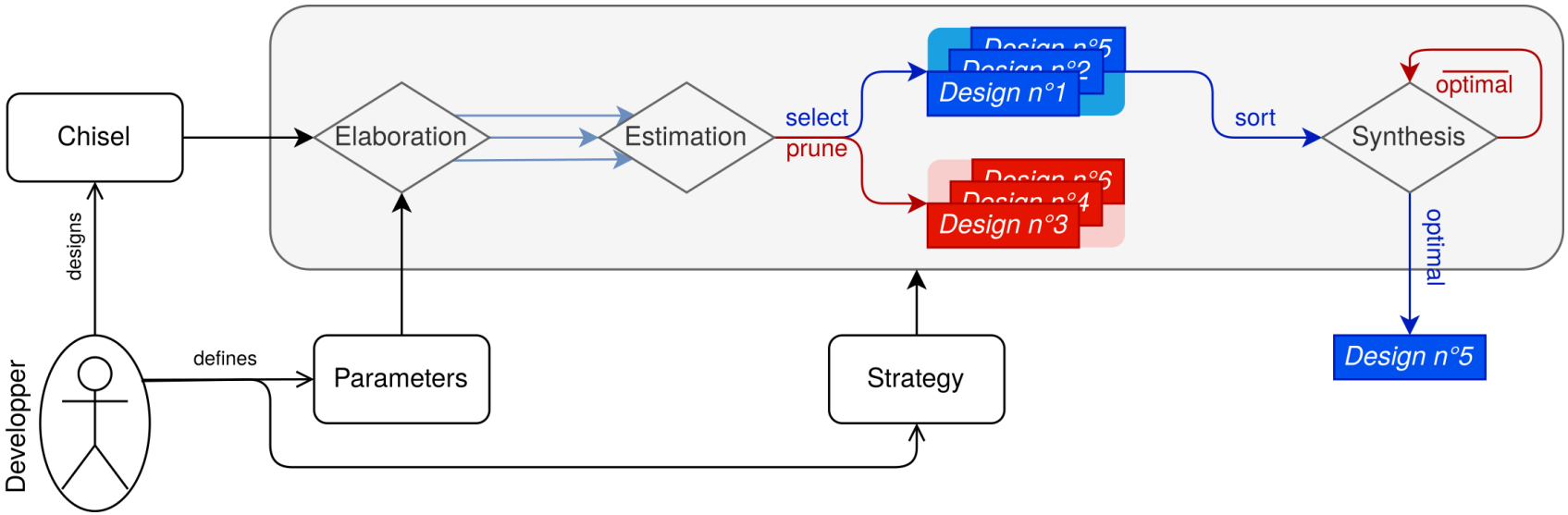


Example of meta exploration strategy

Step 2

Gradient approach and syntheses to find a realistic solution

Proposed Methodologies



Example of meta exploration strategy

Synthesis

Dual-methodology for efficient **HCL-based DSE**...

... but how to demonstrate its **usability**?

Plan

Building a Demonstrator

Technical contribution

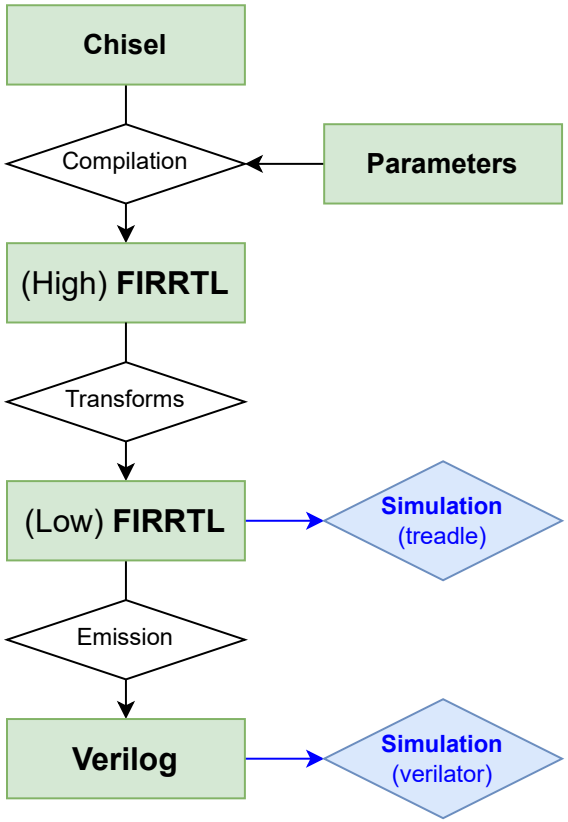
- build a **demonstrator** for the **proposed methodology**
- leverage **Scala features** for expertise-based DSE
- provide a **functional API** for concise definition of design processes
  - focus on providing a **flexible API**
- **open-source** code!

Proposed Framework

QECE (*Quick Exploration using Chisel Estimators*)

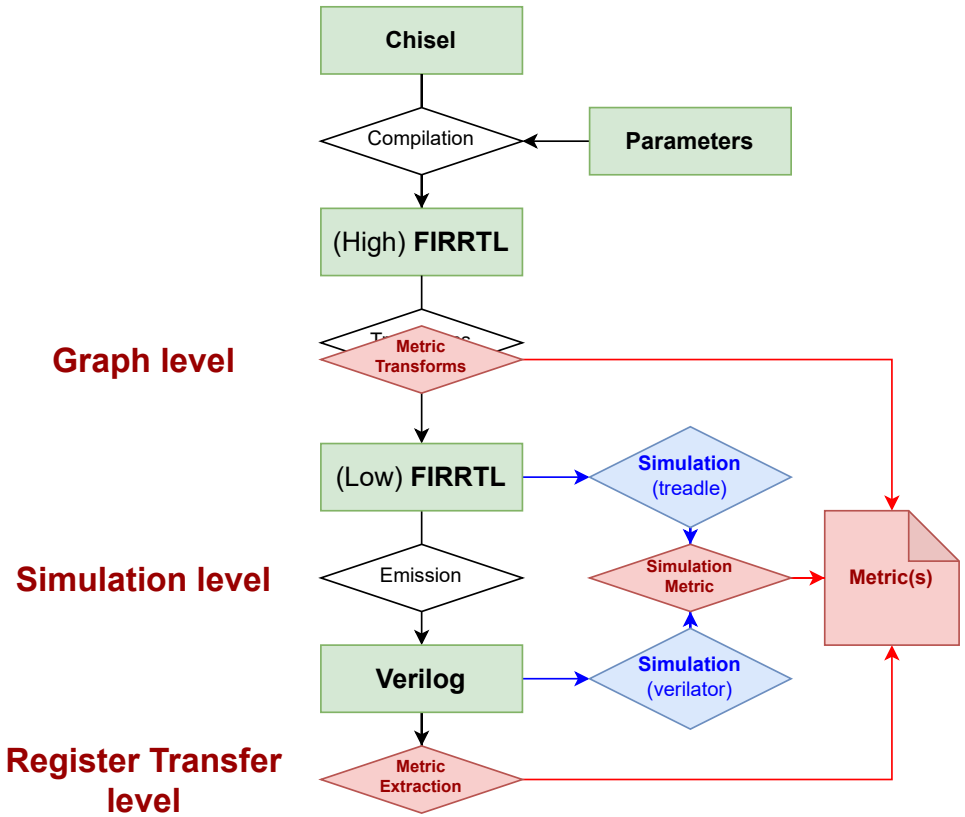
- allows estimations at different **abstraction levels**
- provides a **functional exploration API**
- **built-in libraries** of estimators and strategies
- provided as an **open-source Scala package**
  - can be imported in any **Chisel-based project**
  - provided with an **applicative benchmark**

QECE Structure and API



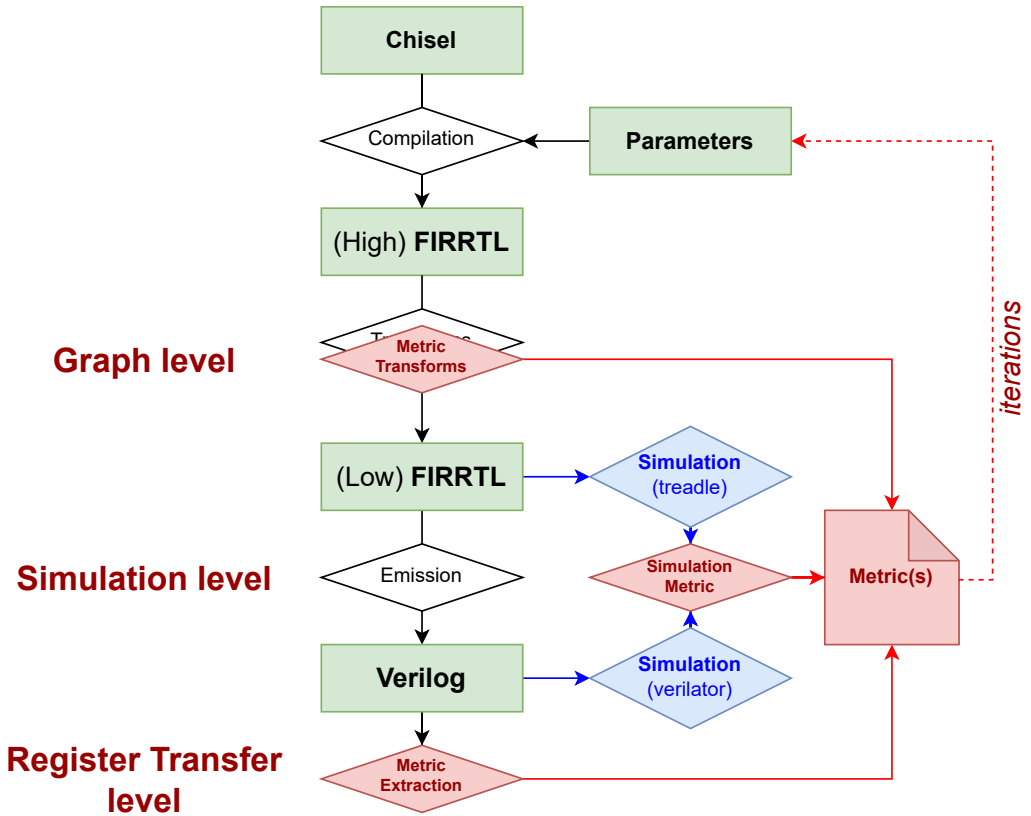
Standard Chisel design flow

QECE Structure and API



Integrating estimators in Chisel

QECE Structure and API



Proposed HCL-based DSE approach

Built-in Estimators

Metrics of interest

- resource usage — *LUT, FF, DSP, BRAM*
- operating frequency — *MHz, ns*
- quality of service — *error rate*
- custom metrics — *latency, throughput, ...*

Built-in Estimators

Metrics of interest

- resource usage — *LUT, FF, DSP, BRAM*
- operating frequency — *MHz, ns*
- quality of service — *error rate*
- custom metrics — *latency, throughput, ...*

Estimation methodologies

- **Graph level**
  - resource usage — *characterized library*
  - custom metrics — *analytical formulas*
- **Simulation level**
  - quality of service — *empirical approach*
- **Register Transfer Level**
  - resource usage & operating frequency — *synthesis results*

Built-in Strategies

Functional exploration

Strategies can be built by composing **basic exploration steps**

Built-in Strategies

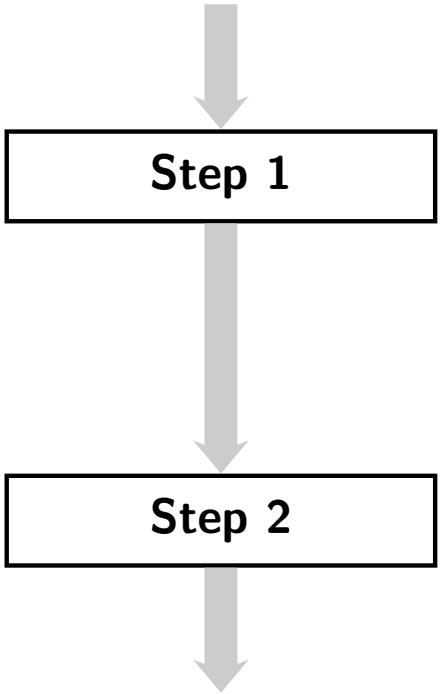
Functional exploration

Strategies can be built by composing **basic exploration steps**

Library of basic steps:

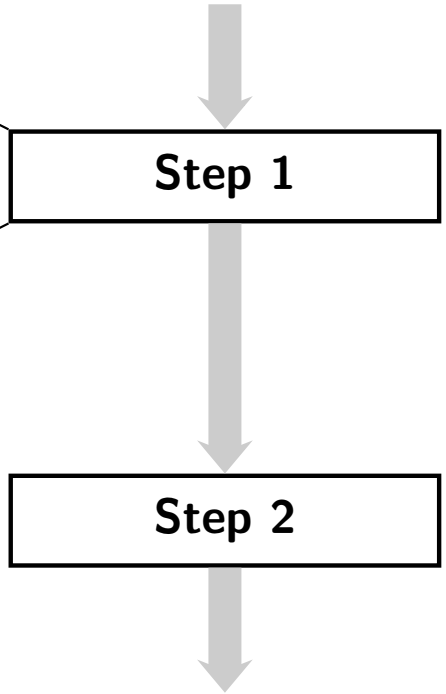
- exhaustive function **map**
- exhaustive **sort**
- exhaustive **prune**
- **gradient**-based sort
- **quick pruning**

Simple Usecase: Demonstrating QECE Expressivity

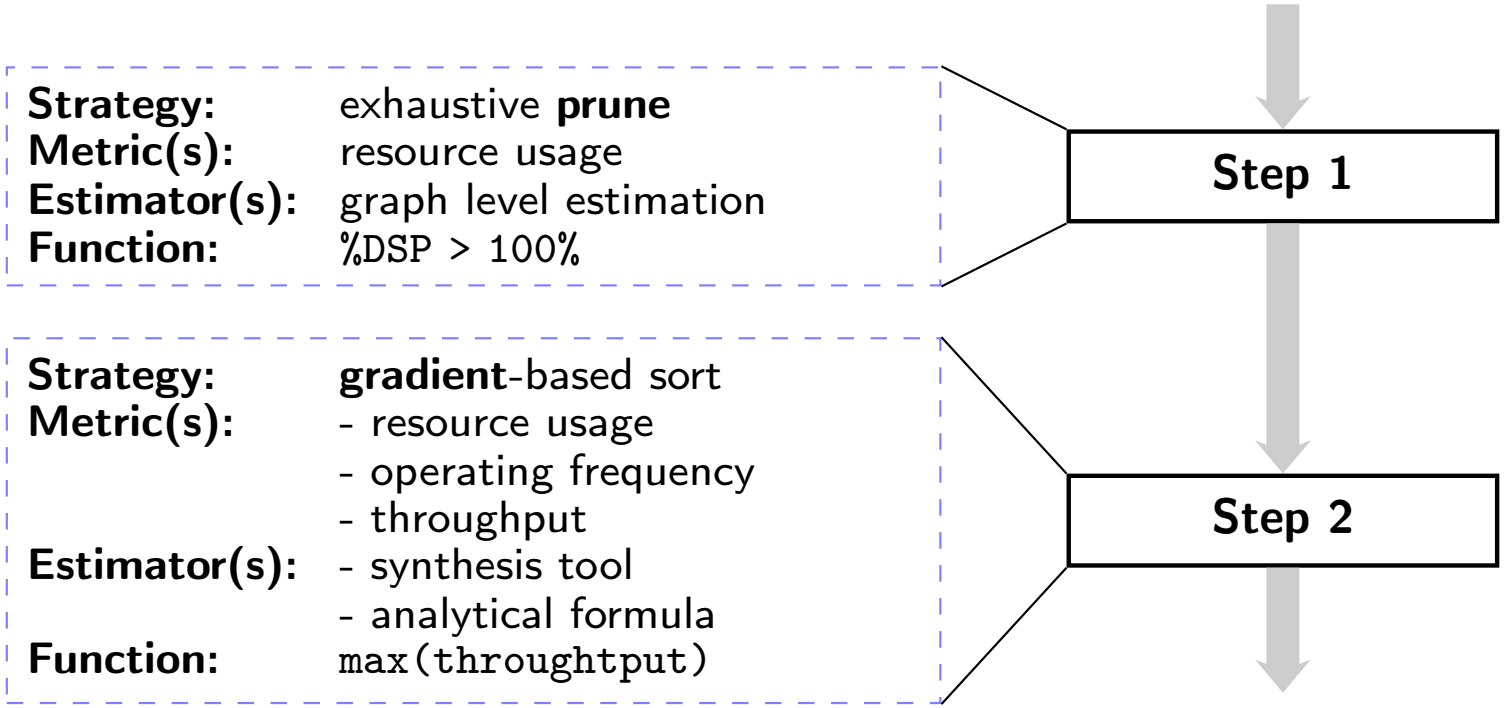


Simple Usecase: Demonstrating QECE Expressivity

Strategy: exhaustive **prune**
Metric(s): resource usage
Estimator(s): graph level estimation
Function: %DSP > 100%

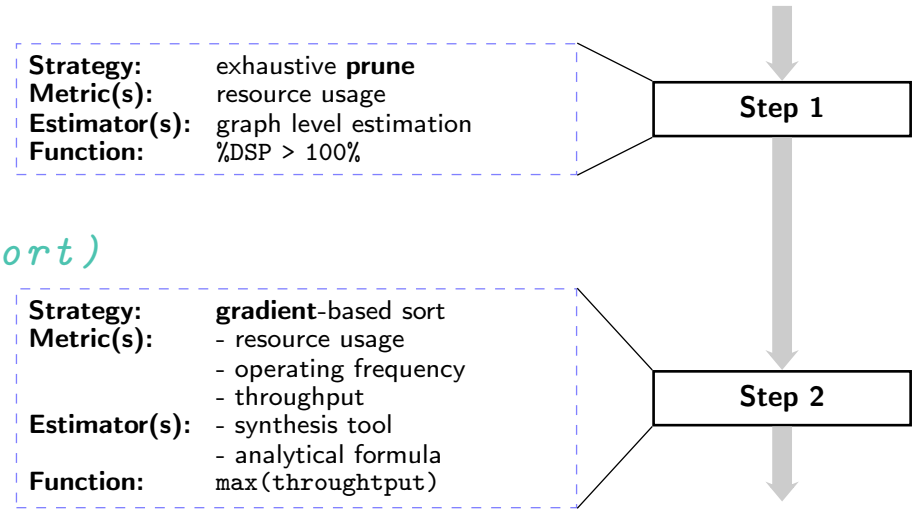


Simple Usecase: Demonstrating QECE Expressivity



Simple Usecase: Demonstrating QECE Expressivity

```
0  val builder  = StrategyBuilder()
1  val strategy = builder.buildStrategy(
2    builder.compose(
3      // first step (prune)
4      builder.prune[GemmModule](
5        tfs    = TransformSeq.resources,
6        func   = m => m("%dsp") > 1.0
7      ),
8      // second step (gradient sort)
9      builder.gradient[GemmModule](
10       tfs    = TransformSeq.synthesis ++
11                throughputFormula,
12       func   = _("throughput"),
13       cmp    = (_ > _),
14       numThread = 4
15     )
16   )
17 )
```



Implementing an exploration strategy with QECE

- only a **few lines of code**
- each step can be **tuned finely**

Proposed Framework

Technical contributions

- **QECE** is proposed as an operational **open source** library[1]
  - embedded **libraries** to build **estimation** and **exploration** processes
  - **complex processes** can be described in a **concise way**
  - article to be submitted to **ACM TODAES** to demonstrate usability
- an **applicative benchmark** has been developed for demonstrations

[1] https://gricad-gitlab.univ-grenoble-alpes.fr/tima/sls/projects/

Plan

Applicative Benchmark

Custom benchmark of **significant FPGA kernel** generators, including:

- **GEMM** (General Matrix Multiply) *Image processing*
- **Black Scholes** computation unit *Finance*

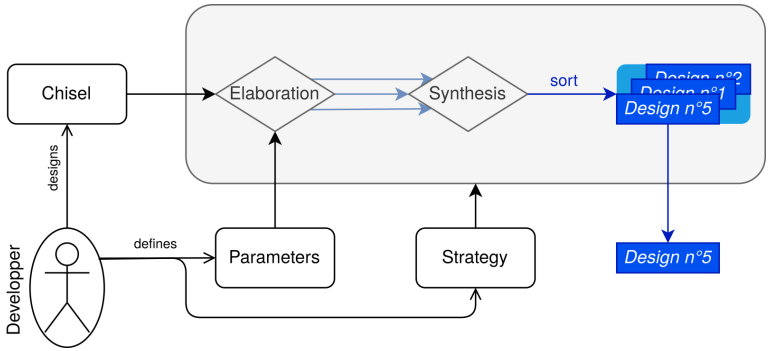
Simple Usecase — Exploring GEMM meta design

Demonstration goal

Compare different **exploration strategies** for a **GEMM-based usecase**:

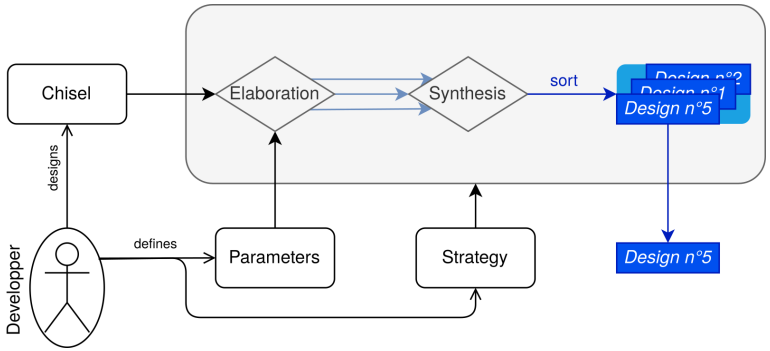
- **FPGA** target board: **Xilinx VC709**
- maximize the **throughput** on the given board

Simple Usecase — Exploring GEMM meta design

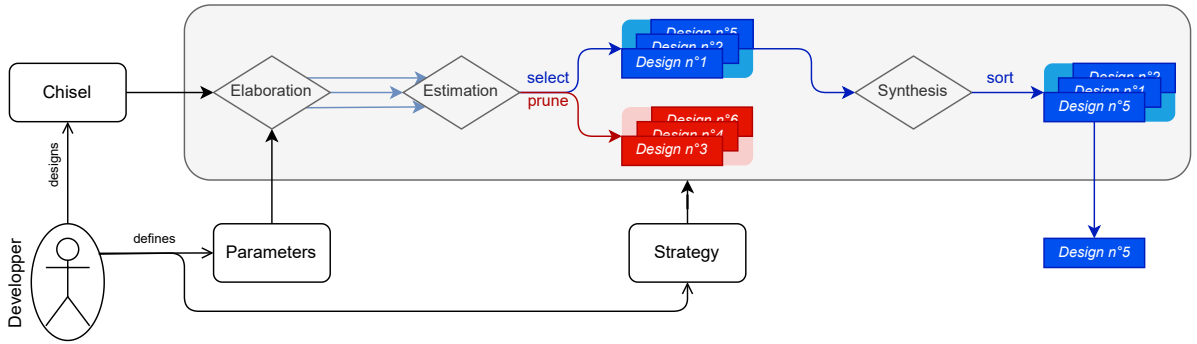


Exhaustive strategy

Simple Usecase — Exploring GEMM meta design

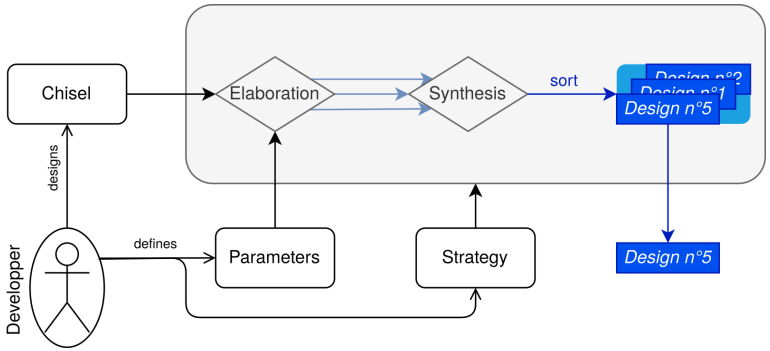


Exhaustive strategy

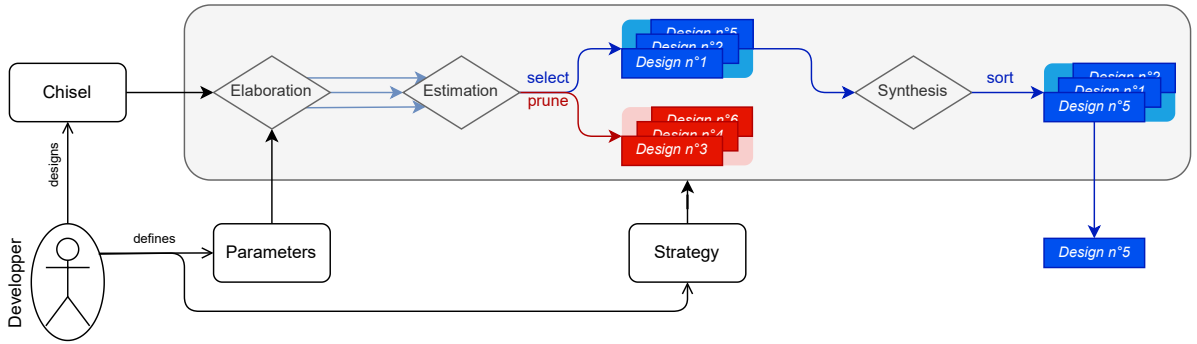


Pruning strategy

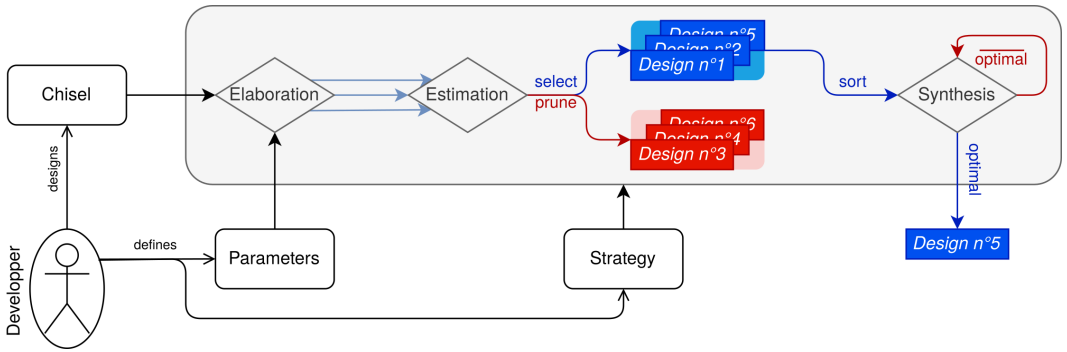
Simple Usecase — Exploring GEMM meta design



Exhaustive strategy



Pruning strategy



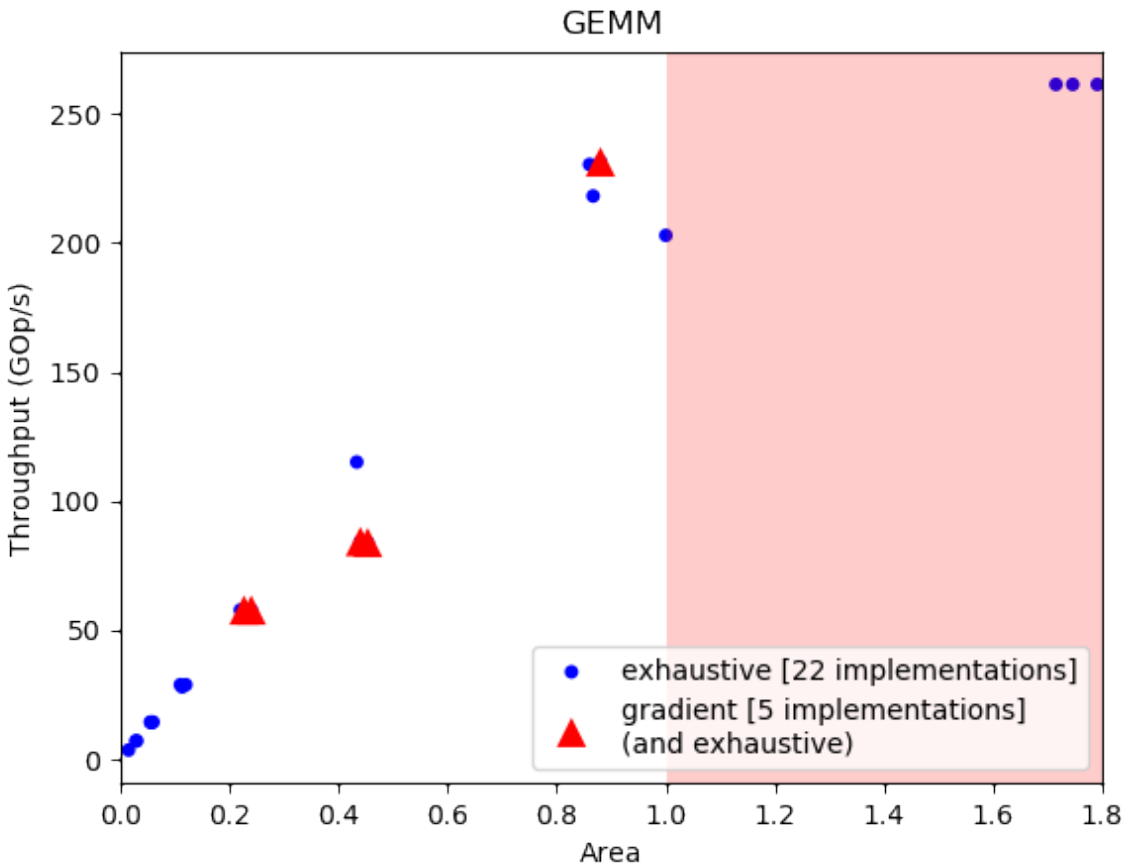
Gradient strategy

Simple Usecase — Exploring GEMM meta design

Strategy	**Best throughput**	**#(space)**	**#synth** (#timeout)	**Time**	**Speed-up**
Exhaustive	*231.334 GOp/s*	41	*41 (19)*	*13h51m56s*	-
Pruning	231.334 GOp/s		26 (7)	08h52m00s	×1.5
Gradient	231.334 GOp/s		6 (1)	03h21m06s	×4.1

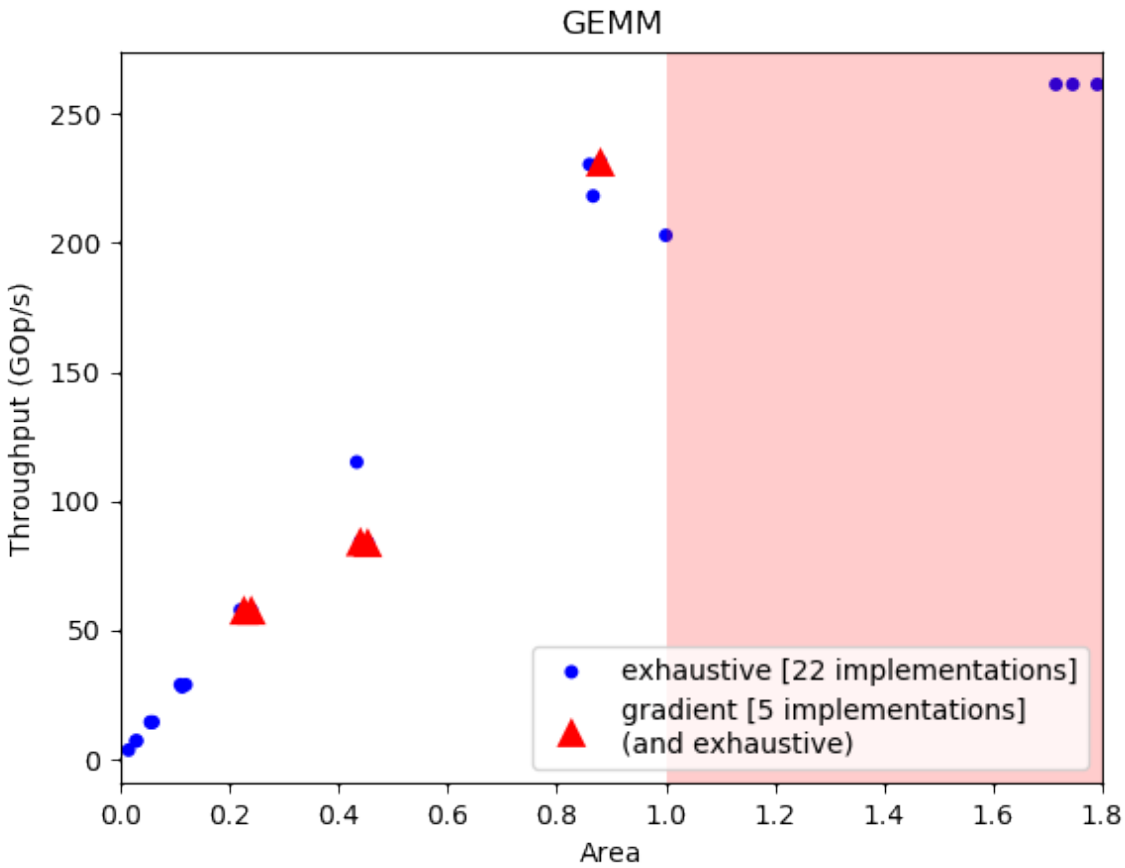
Simple Usecase — Exploring GEMM meta design

Strategy	Best throughput	#(space)	#synth (#timeout)	Time	Speed-up
Exhaustive	*231.334 GOp/s*	41	*41 (19)*	*13h51m56s*	-
Pruning	231.334 GOp/s		26 (7)	08h52m00s	×1.5
Gradient	231.334 GOp/s		6 (1)	03h21m06s	×4.1



Simple Usecase — Exploring GEMM meta design

Strategy	Best throughput	#(space)	#synth (#timeout)	Time	Speed-up
Exhaustive	*231.334 GOp/s*		*41 (19)*	*13h51m56s*	-
Pruning	231.334 GOp/s	41	26 (7)	08h52m00s	×1.5
Gradient	231.334 GOp/s		6 (1)	03h21m06s	×4.1



Results

- reduced the **number of syntheses**
- found the **same optimum**

Exploring Black Scholes Implementations

Black Scholes Model

Black Scholes model is used to estimate the **price of an option**:

- based on the **Monte Carlo** method
- **accuracy** depends on the **number of samples**
- **latency** depends on the **number of parallel cores**
- need for an **applicative model** to define the **quality of service**

```
class BlackScholes(
    @resource @qos @linear(8, 32)  dynamic: Int,
    @resource @qos @linear(8, 32)  precision: Int,
    @qos @pow2(5, 10)              nbIteration: Int,
    @qos @pow2(1, 6)               nbEuler: Int,
    @resource @pow2(2, 10)         nbCore: Int
) extends Module with Explorable {...}
```

Exploring Black Scholes Implementations

Black Scholes Model

Black Scholes model is used to estimate the **price of an option**:

- based on the **Monte Carlo** method
- **accuracy** depends on the **number of samples**
- **latency** depends on the **number of parallel cores**
- need for an **applicative model** to define the **quality of service**

Application Specific Exploration

Two concerns for the exploration:

- respect an **error rate** threshold — *i.e.* $< 5\%$
- select the implementation with the **best throughput**

Exploring Black Scholes Implementations

Black Scholes Model

Black Scholes model is used to estimate the **price of an option**:

- based on the **Monte Carlo** method
- **accuracy** depends on the **number of samples**
- **latency** depends on the **number of parallel cores**
- need for an **applicative model** to define the **quality of service**

Application Specific Exploration

Two concerns for the exploration:

- respect an **error rate** threshold — *i.e.* $< 5\%$
- select the implementation with the **best throughput**

5 generation parameters $\Rightarrow$ **202,500 implementations**

$\hookrightarrow$ need for an **intelligent strategy** to explore them

Exploring Black Scholes Implementations

Implemented Exploration Strategy

Based on **5 sequential steps**, including:

Exploring Black Scholes Implementations

Implemented Exploration Strategy

Based on **5 sequential steps**, including:

- quality of service based **pruning** $\simeq$ 37h ($\approx 22,500$ simulations)
- synthesis based sorting $\simeq$ 30 min (only **27 actual syntheses**)

Exploring Black Scholes Implementations

Implemented Exploration Strategy

Based on **5 sequential steps**, including:

- quality of service based **pruning** $\simeq$ 37h ($\approx 22,500$ simulations)
- synthesis based sorting $\simeq$ 30 min (only **27 actual syntheses**)

Rank	Parameters	Error	Throughput ($est.s^{-1}$)	Area		Frequency
				Max %	Resource	
1	[12, 21, 64, 2, 64]	5.34%	125.06	25%	DSP	250.13 MHz
2	[12, 20, 64, 2, 64]	4.6%	125.06	25%	DSP	250.13 MHz
3	[12, 22, 64, 2, 64]	6.54%	125.03	25%	DSP	250.06 MHz
4	[13, 21, 64, 2, 64]	6.06%	125.03	25%	DSP	250.06 MHz
5	[12, 22, 64, 2, 32]	5.34%	62.53	12.5%	DSP	250.13 MHz

Synthesis on the Experiments

Synthesis

QECE usability has been demonstrated on **multiple usecases**:

Synthesis on the Experiments

Synthesis

QECE usability has been demonstrated on **multiple usecases**:

- different **exploration strategies** have been compared

Synthesis on the Experiments

Synthesis

QECE usability has been demonstrated on **multiple usecases**:

- different **exploration strategies** have been compared
- various **concerns** and **objectives** have been considered

Synthesis on the Experiments

Synthesis

QECE usability has been demonstrated on **multiple usecases**:

- different **exploration strategies** have been compared
- various **concerns** and **objectives** have been considered
- the sources for the experiments can be **found online**

Synthesis on the Experiments

Synthesis

QECE usability has been demonstrated on **multiple usecases**:

- different **exploration strategies** have been compared
- various **concerns** and **objectives** have been considered
- the sources for the experiments can be **found online**

Doing so, we also demonstrated the **conciseness** of the approach

Plan

Contributions

Conceptual contributions

- HCL-based design flow based on a **dual-methodology**
- **Functional approach** for design space exploration
- **Productivity** increases thanks to:
  - HCLs, which improve **reusability**
  - QECE, with **concise and powerful DSE processes**

Contributions

Conceptual contributions

- HCL-based design flow based on a **dual-methodology**
- **Functional approach** for design space exploration
- **Productivity** increases thanks to:
  - HCLs, which improve **reusability**
  - QECE, with **concise and powerful DSE processes**

Technical contributions

- **PoC** framework: **QECE** (*Quick Exploration using Chisel Estimators*)
  - **built-in libraries** of estimators and exploration strategies
  - allows defining **complex strategies** in a **concise way**
- Demonstration benchmark: set of **FPGA kernel generators**
- Both projects are **open-sourced** in TIMA/SLS gitlab[1]

[1] https://gricad-gitlab.univ-grenoble-alpes.fr/tima/sls/projects/

Perspectives

Estimation features

- improve the provided **estimators**
  - better timing estimators
  - machine learning approaches
- introduce **other metrics & estimation methodologies**
  - **routability**, **memory usage** and other **technological concerns**
  - provide **multiple ways** to estimate the metrics
- provide direct **feedback and hints** to the users
  - quick feedback on **routability** and **resource usage**
  - estimate of the **critical areas** of a design

Perspectives

Exploration features

- improve the **library of exploration strategies**
  (*e.g.* reinforcement learning, genetic algorithms, ...)
- provide **visualization models** for the users
- allow **hierarchical explorations**
  - could be used to build **Systems on Chip**
  - would cope with our **flexible approach**
- demonstrate on a **more realistic usecase**
  - could consider an industrial usecase
  - explore our custom **MultiLayer Perceptron** generator

Publications

International Conferences and Workshops:

- **Chisel Usecase: Designing General Matrix Multiply for FPGA**

  Bruno Ferres, Olivier Muller, Frédéric Rousseau. In *Proc. of the* 16^{th} *International Symposium on Applied Reconfigurable Computing* (**ARC2020**), Toledo, Spain, 2020.

- **Integrating Quick Resource Estimators in Hardware Construction Framework for Design Space Exploration**

  Bruno Ferres, Olivier Muller, Frédéric Rousseau. In *Proc. of the* 32^{nd} *International Workshop on Rapid System Prototyping* (**RSP'21**).

Journal Articles (*to be submitted*):

- **Chisel Framework for Flexible Design Space Exploration through a Functional Approach**

  Bruno Ferres, Olivier Muller, Frédéric Rousseau. In *ACM Transactions on Design and Automation of Electronic Systems* (**TODAES**), 2022.

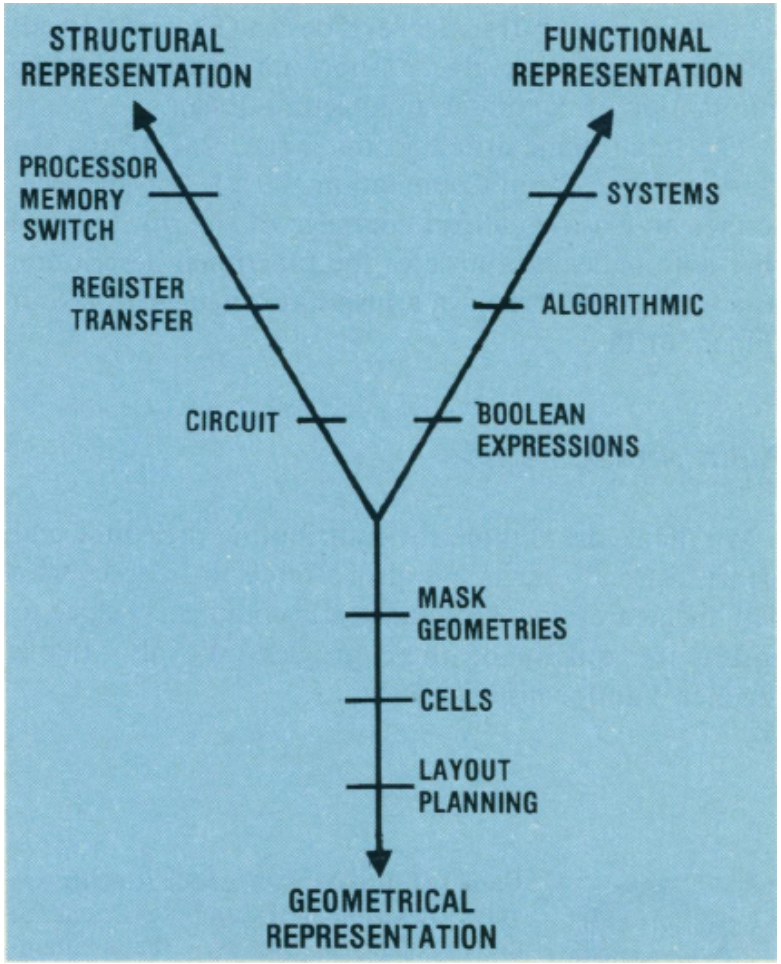
Thank you for your attention...

Thank you for your attention...

...now it's **question time**!

APPENDICES

Abstraction Levels for Digital Design



Gajski-Khun chart [GK83][1]

[1] Gajski and Kuhn, "New VLSI Tools", in *Computer*, 1983

State of the Art: DSE Approaches and Tools

- No other approach proposing a **flexible** and **functional approach** for defining **design space exploration strategies**
- Many approaches for HLS-based DSE (*e.g.* [PBMR14][1]) and DSL-based DSE (*e.g.* [NKO19][2])
- **Dovado** leverages HDL parameters for exploration ([PCS21][3])

[1] Prost-Boucle *et al.*: "Fast and standalone Design Space Exploration for High-Level Synthesis under resource constraints.", in *Journal of Systems Architecture*, 2014

[2] Nardi *et al.*: "Practical Design Space Exploration.", in *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2019

[3] Paletti *et al.*: "Dovado: An Open-Source Design Space Exploration Framework.", in *International Parallel and Distributed Processing Symposium Workshops*, 2021

State of the Art: Estimation and Exploration

- a lot of research on **estimation methodologies** (*e.g.* [SJ08][1])
- we only consider **sequential strategies** at the moment ([SR18][2])
- different types of strategies considered ([SW20][3])
  - **meta heuristics**, **dedicated heuristics** and **supervised learning**
  - **graph analysis** — cannot be applied directly with HCLs

[1] Schumacher *et al.*: "Fast and accurate resource estimation of RTL-based designs targeting FPGAS.", in *International Conference on Field Programmable Logic and Applications*, 2008

[2] Shatanaa *et al.*: "Design Space Exploration for Architectural Synthesis — A Survey.", in *Recent Finding in Intelligent Computing Techniques*, 2018

[3] Schaffer *et al.*: "High-Level Synthesis Design Space Exploration: Past, Present, and Future.", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020

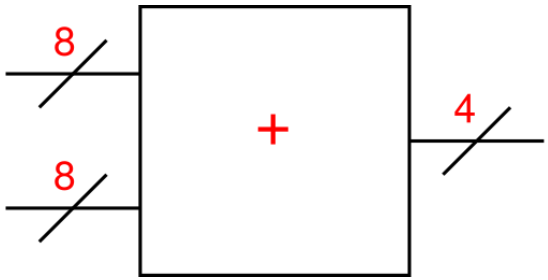
What about SystemVerilog?

- a lot of interesting features (*POO*, *UVM*, ...) for **simulation purposes**
- some interesting features for **circuit description**:
  - more complex **data types**
    (`enum` and `struct`, multidimensional arrays, `logic` registers, ...)
  - **behavioral indications**:
    `always_comb`, `always_ff` and `always_latch`
    $\Rightarrow$ useful for **readability** and **warnings**
  - **advanced interface** definition with custom bundles
  - verilog is a **subset** of SystemVerilog
    $\Rightarrow$ both are **Hardware Description Languages**
    $\Rightarrow$ a lot of features are actually from SystemVerilog
    (custom types, 2D arrays, ...)
- ...but no powerful features such as **POO** or **functional programming** for **circuit description**

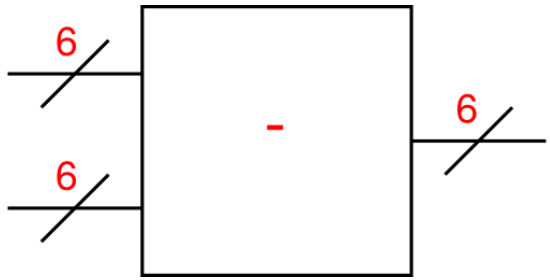
What about SystemC?

- based on a **C++ library**
- used at **system level** to model the **behaviour** of **complex systems**
  - a lot **simulation features** at different **levels** (*register transfer level*, *transaction level*, *platform level*)
  - useful for **co-design** and **virtual prototyping**
- **HLS tools** can generate components from **algorithmic descriptions**
- power features (*POO*, *templates*, ...) for **simulation** and **prototyping**
  - but limited performances on **circuit descriptions**
  - semantic and approach are **tightly coupled to simulation**

Generic Implementation with Verilog




```
module adder
    #(parameter IN_WIDTH=8,
      parameter OUT_WIDTH=8)
    (input [IN_WIDTH-1:0] op1,
     input [IN_WIDTH-1:0] op2,
     output [OUT_WIDTH:0] out);
    localparam MAX = 1 << OUT_WIDTH - 1;
    wire [IN_WIDTH:0] tmp;
    tmp = op1 + op2
    out = tmp > MAX ?
        MAX : tmp[OUT_WIDTH-1:0] ;
end module
```




```
module sub
    #(parameter IN_WIDTH=8,
      parameter OUT_WIDTH=8)
     (input [IN_WIDTH-1:0] op1,
      input [IN_WIDTH-1:0] op2,
      output [OUT_WIDTH-1:0] out);
    out = op1 - op2;
end module
```

Some Chisel-Based Projects[1]

- **Rocket Chip Generator** — *RISC-V in-order generator*
- **BOOM**[2] — *RISC-V out-of-order generator*
- **Edge TPU (from Google)** — *AI Inference Accelerator*
- **DANA** — *Multilayer Perceptron Accelerator for Rocket*
- **Gemmini** — *Systolic-array Accelerator Generator*
- ...

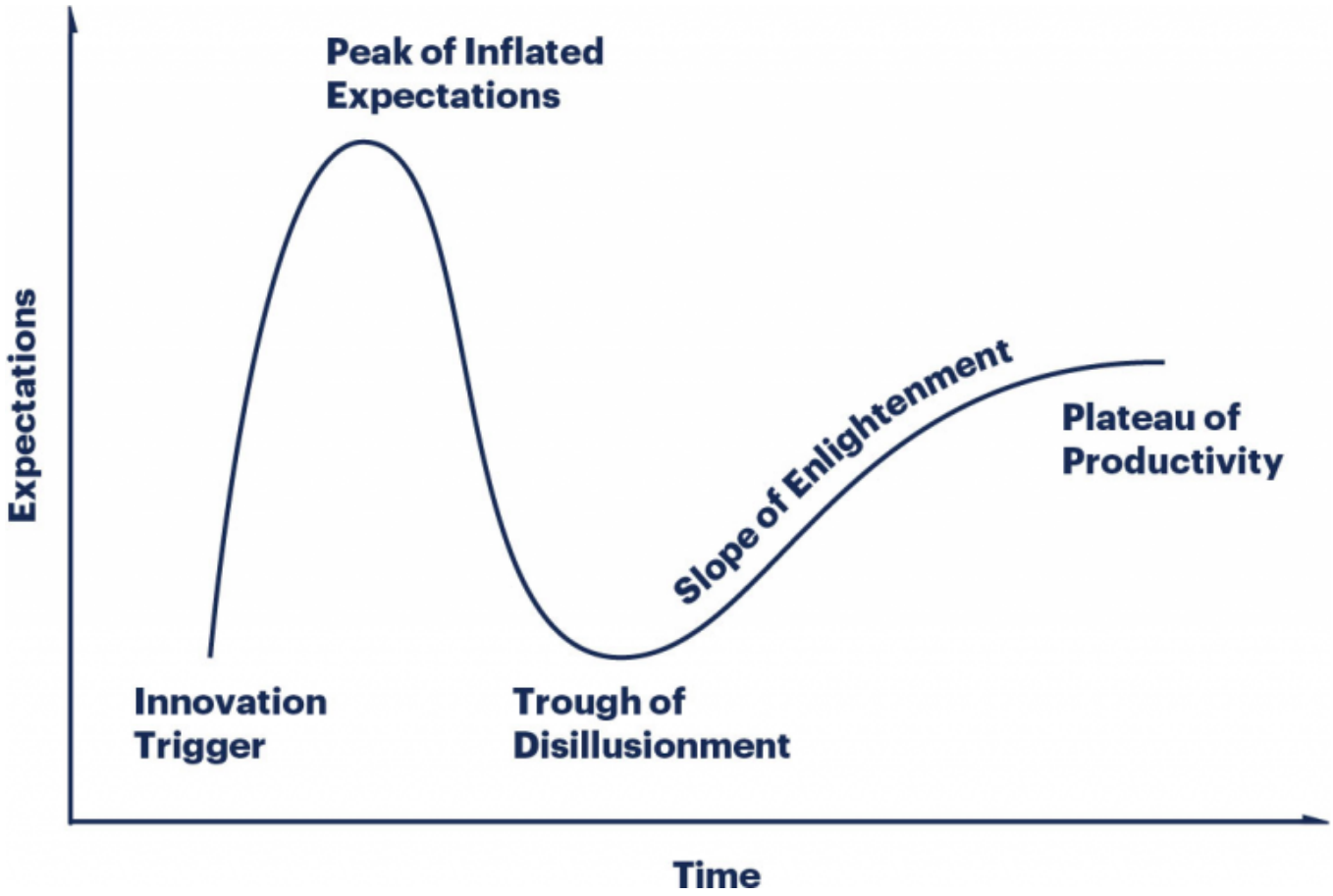
[1] From https://www.chisel-lang.org/community.html
[2] Berkeley Out-of-order Machine

Building a Functional Block in Chisel

```
class CustomBlock[T <: Data](gen: T)(
    inputBitwidth:  Int,
    outputBitwidth: Int,
    func:           (T, T) => T
) extends Module {
    val io = IO(new Bundle {
        val op1 = Input(UInt(inputBitwidth.W))
        val op2 = Input(UInt(inputBitwidth.W))
        val out = Input(UInt(outputBitwidth.W))
    })
    io.out := func(
        io.op1.asTypeOf(gen),
        io.op2.asTypeOf(gen)
    ).asTypeOf(io.out)
}

val adder   = new CustomBlock(SInt(8.W))(8, 4, _ + _)
val sub     = new CustomBlock(UInt(6.W))(6, 6, _ - _)
```

Hardware Construction Languages and Productivity



Hype Cycle from Gartner[1]

[1] From https://www.gartner.com

Hardware Construction Languages and Productivity

Rough approximation of productivity gain, based on our personal experiences

	Methodology		
Criterion	HDL	HLS	**HCL**
Learning	weeks	days	weeks (days)
Mastering	months	months	months (months)
Design	days	hours	days
Optimization	days	days	days
Debug	days	days	days
Reuse	days	days	hours

Feedback on the use of Hardware Construction Languages

3 years expertise on Chisel usage, with:

- **two PhD students**
- **four engineer interns**

HCL-based DSE: Motivations

Exploring an accelerator generator

Standard DSE methodologies not applicable for HCLs:

- HLS-based methodologies use **inferences**
- HDL-based methodologies does not consider **high level parameters**

HCL-based DSE: Motivations

Exploring an accelerator generator

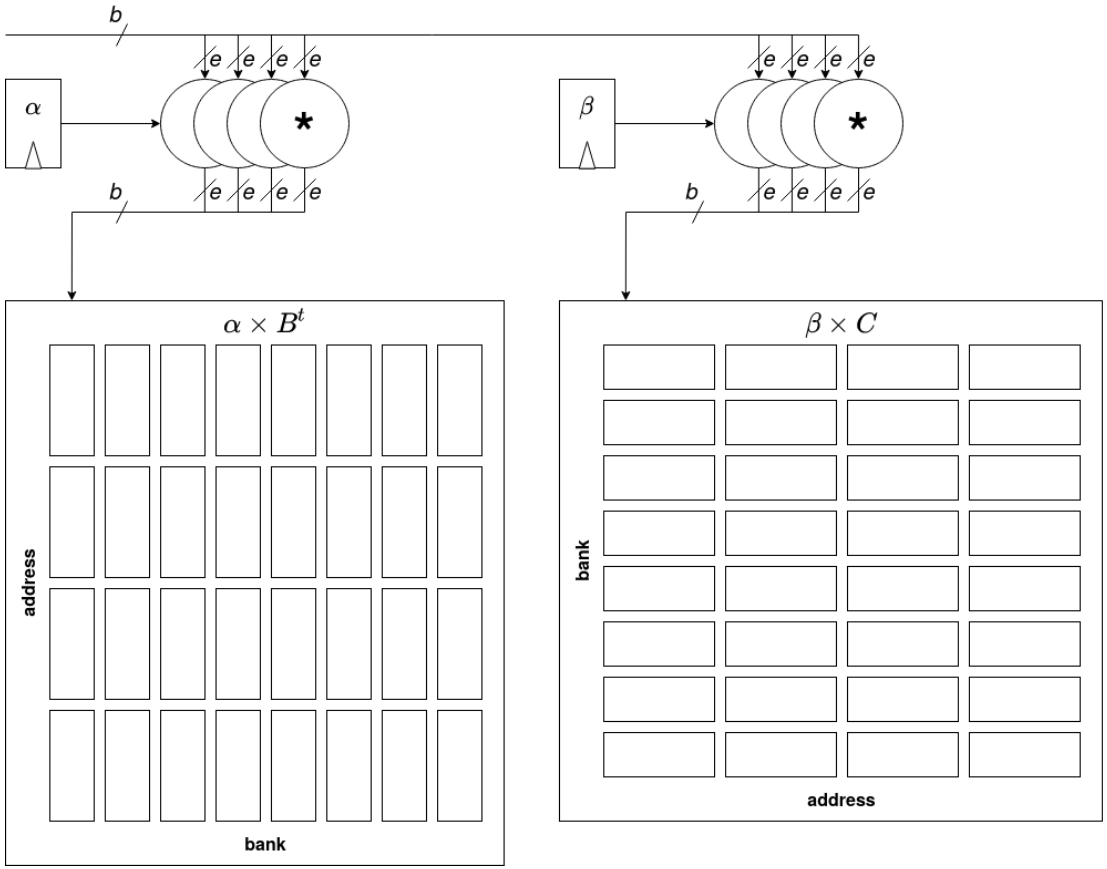
Standard DSE methodologies not applicable for HCLs:

- HLS-based methodologies use **inferences**
- HDL-based methodologies does not consider **high level parameters**

Purpose of an HCL-based DSE methodology

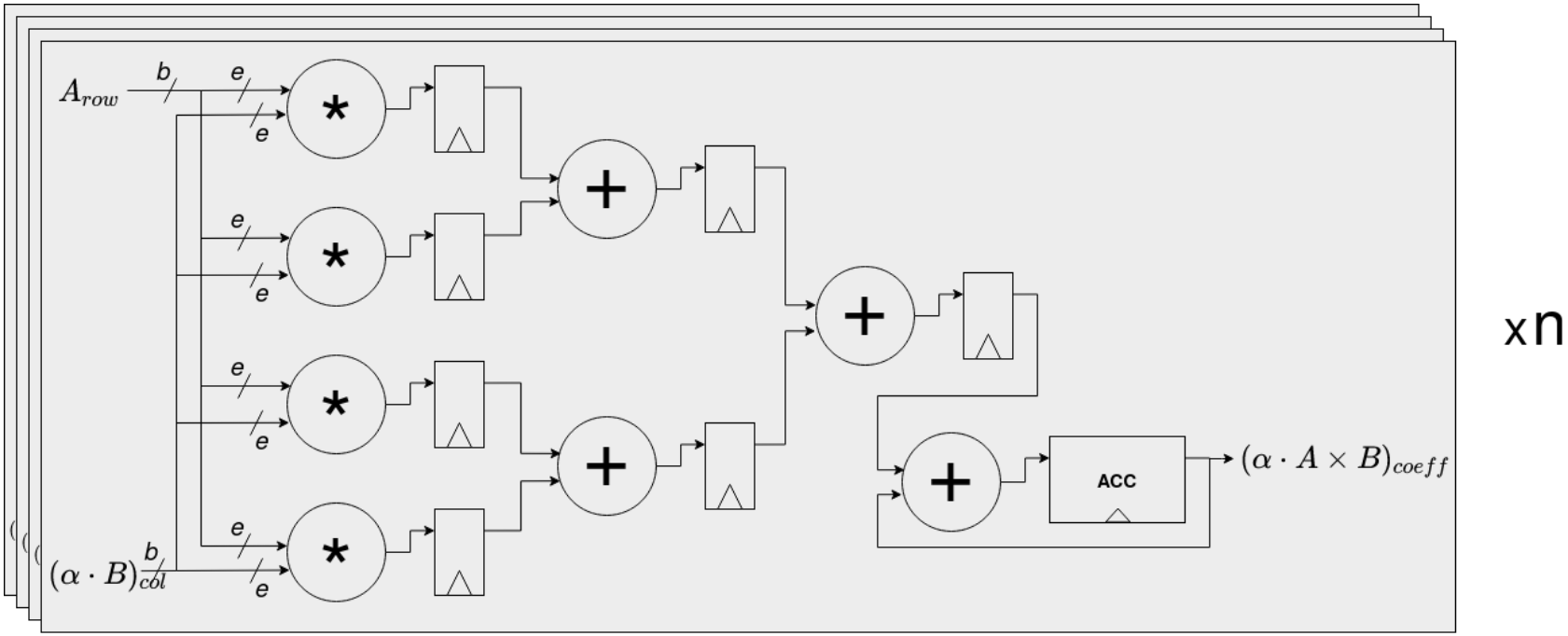
- shall take advantage of the **HCL features**
  - module **generation**
  - **high-level** programming features
- **expertise-based** approach

GEMM Architecture



Chosen memory architecture

GEMM Architecture



Chosen computation unit architecture

Application: Fast Fourier Transform

```
class FftModule(
    @pow2(1, 10) parallelism:   Int,
    @enum(32)    elementWidth:  Int,
    @enum(SIZE)  size:          Int
) extends Module { ... }
```

Exposing **Fast Fourier Transform** design space

Application: Fast Fourier Transform

Size	Strategy	Best throughput	#(space)	#synth (#timeout)	Time	Speed-up
128	*Exhaustive*	*1.767 Tb/s*	7	*7 (0)*	*00h22m45s*	-
	Pruning	1.767 Tb/s		7 (0)	00h24m14s	×0.94
	Gradient	1.767 Tb/s		3 (0)	00h19m51s	×1.15
512	*Exhaustive*	*5.479 Tb/s*	9	*9 (0)*	*02h11m51s*	-
	Pruning	5.479 Tb/s		9 (0)	03h17m52s	×0.66
	Gradient	5.479 Tb/s		3 (0)	02h18m29s	×0.95

Application: Fast Fourier Transform

Size	Strategy	Best throughput	#(space)	#synth (#timeout)	Time	Speed-up
128	*Exhaustive*	*1.767 Tb/s*	7	*7 (0)*	*00h22m45s*	-
	Pruning	1.767 Tb/s		7 (0)	00h24m14s	×0.94
	Gradient	1.767 Tb/s		3 (0)	00h19m51s	×1.15
512	*Exhaustive*	*5.479 Tb/s*	9	*9 (0)*	*02h11m51s*	-
	Pruning	5.479 Tb/s		9 (0)	03h17m52s	×0.66
	Gradient	5.479 Tb/s		3 (0)	02h18m29s	×0.95

Results

- found the **same optimum**

Application: Fast Fourier Transform

Size	Strategy	Best throughput	#(space)	#synth (#timeout)	Time	Speed-up
128	*Exhaustive*	*1.767 Tb/s*	7	*7 (0)*	*00h22m45s*	-
	Pruning	1.767 Tb/s		7 (0)	00h24m14s	×0.94
	Gradient	1.767 Tb/s		3 (0)	00h19m51s	×1.15
512	*Exhaustive*	*5.479 Tb/s*	9	*9 (0)*	*02h11m51s*	-
	Pruning	5.479 Tb/s		9 (0)	03h17m52s	×0.66
	Gradient	5.479 Tb/s		3 (0)	02h18m29s	×0.95

Results

- found the **same optimum**
- but only consider a **small design space**
  - only **one meaningful parameter** (*parallelism*)
  - hence **complex strategies** are **slower**

Application: Fast Fourier Transform

Size	Strategy	Best throughput	#(space)	#synth (#timeout)	Time	Speed-up
128	*Exhaustive*	*1.767 Tb/s*	7	*7 (0)*	*00h22m45s*	-
	Pruning	1.767 Tb/s		7 (0)	00h24m14s	×0.94
	Gradient	1.767 Tb/s		3 (0)	00h19m51s	×1.15
512	*Exhaustive*	*5.479 Tb/s*	9	*9 (0)*	*02h11m51s*	-
	Pruning	5.479 Tb/s		9 (0)	03h17m52s	×0.66
	Gradient	5.479 Tb/s		3 (0)	02h18m29s	×0.95

Results

- found the **same optimum**
- but only consider a **small design space**
  - only **one meaningful parameter** (*parallelism*)
  - hence **complex strategies** are **slower**
- need to compare different *element widths* and *problem sizes*
  - need an **applicative model** to do so!

Meta Exploration Strategy for Black Scholes

```
class BlackScholes(
    @resource @qos @linear(8, 32)  dynamic: Int,
    @resource @qos @linear(8, 32)  precision: Int,
    @qos @pow2(5, 10)              nbIteration: Int,
    @qos @pow2(1, 6)               nbEuler: Int,
    @resource @pow2(2, 10)         nbCore: Int
) extends Module with Explorable {...}
```

Exposed design space

Meta Exploration Strategy for Black Scholes

```
val builder  = StrategyBuilder()
val strategy = builder.buildStrategy(
  context.quickPrune[BlackScholes](
    QualityOfService.simulation,
    _.error > 0.05,
    metric = Some(new qos)
  ),
  context.reduceDimension[BlackScholes](new resource, true),
  context.map[BlackScholes](Transforms.latency),
  context.sort[BlackScholes](<@TransformSeq.empty,
    m => m("dynamic") + m("precision") + m("nbCore"),
    (_ < _)
  ),
  context.gradient[BlackScholes](
    TransformSeq.synthesis ++ Transforms.throughput
    func = _("throughput"),
    cmp = (_ > _)
  )
)
```

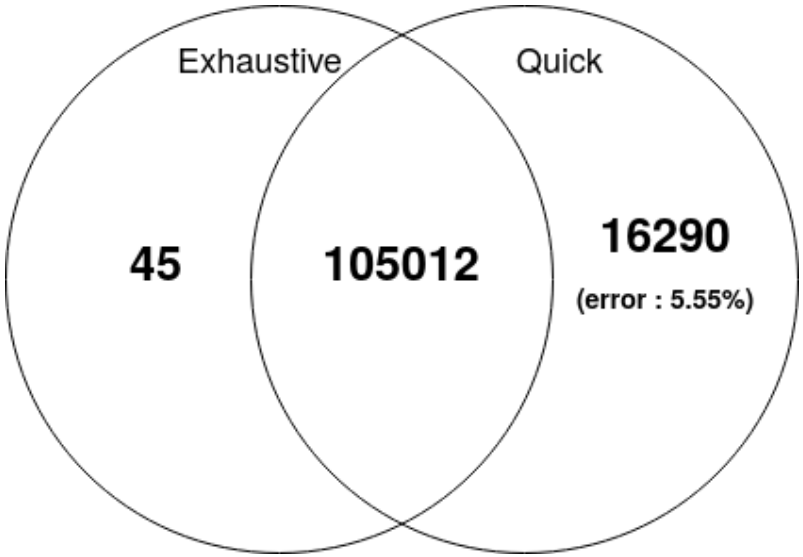
Expertise-based exploration strategy

Meta Exploration Strategy for Black Scholes

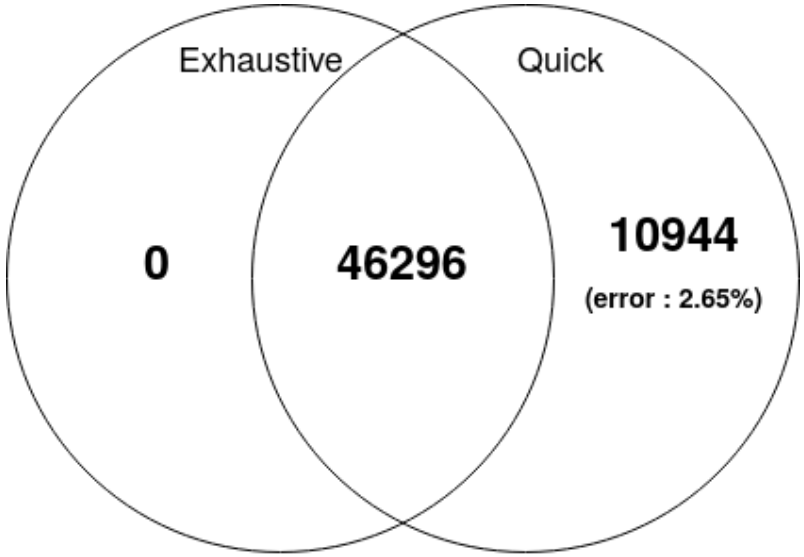
Error threshold	Number of impl.	Strategy	Exploration time	Speed-up
5%	202500	Exhaustive pruning	46h29m19s	-
		Quick pruning	32h59m29s	$\times$1.40
2%	202500	Exhaustive pruning	46h21m42s	-
		Quick pruning	48h46m53s	$\times$0.95

Comparing the pruning strategies over Black Scholes kernels

Meta Exploration Strategy for Black Scholes



5% error threshold



2% error threshold

Comparing the accuracy of the pruning strategies on Black Scholes