A Fast and Stand-alone HLS Methodology for Hardware Accelerator Generation Under Resource Constraints

Adrien Prost-Boucle, Olivier Muller, Frédéric Rousseau

TIMA Laboratory – CNRS/Grenoble-INP/UJF
46 Avenue Félix Viallet, 38000 Grenoble, France

Organization of the presentation

Use of FPGA technologies

- Advantages of FPGA solutions (vs CPU/GPU):

  - High performance
  - Low consumption
  - Massive and fine-grain parallelism
  - High optimization level

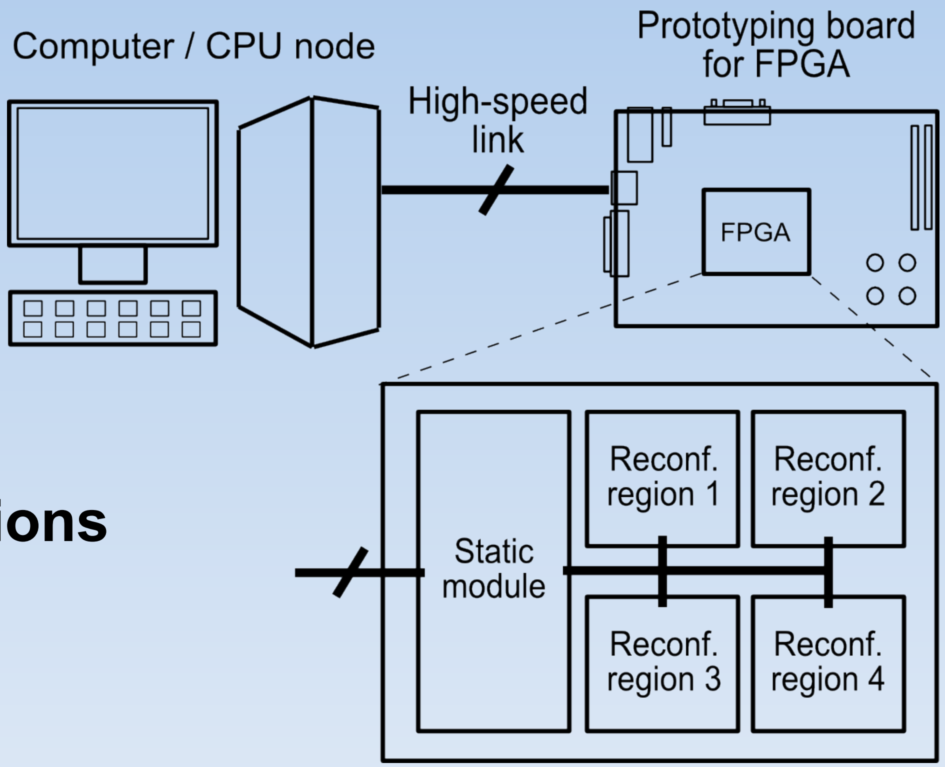
- Used in a broad range of application fields

- But difficult to program.

Our context

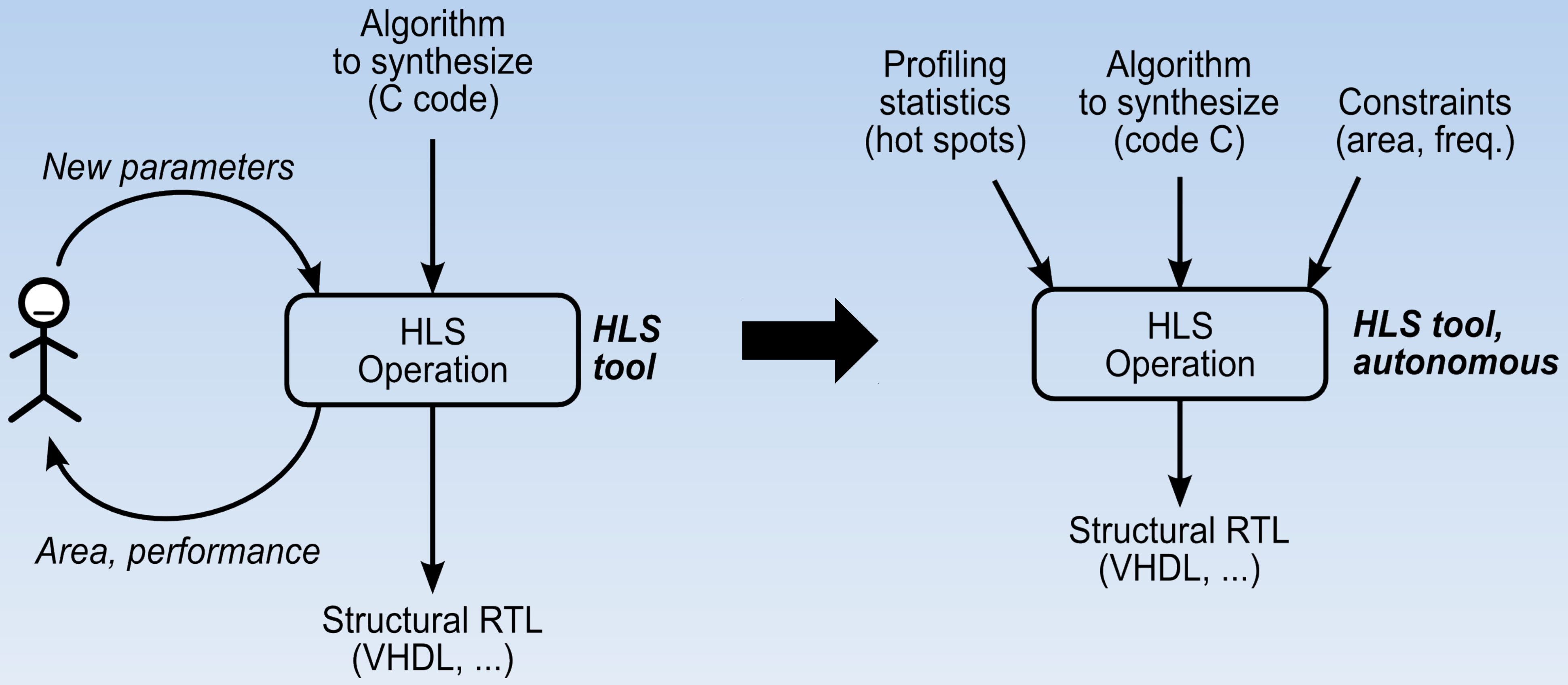
HPC with hardware accelerators:
- Several accelerated modules
- Frequent evolution
- Users have low FPGA expertise

Hardware target :
- CPU + FPGA
- Several reconfigurable regions



Needed design flow for accelerators:
- Fast
- DSE: find an efficient solution
- Respect given resource constraints
- Automated (compilation-like)

Current HLS flow... and needed HLS flow



Related works

Context:
Automatic & fast DSE for HLS

- **Exploration with genetic algorithms**
- **Altera: OpenCL-based flow**
- **Design-Trotter (ref. 8 in abstract)**

  - Limited scalability
  (ROM-based FSM, exhaustive BB implem)
  - Other works, using GAUT:
  DSE from latency constraint

Design Space Exploration

Exploration priorities:

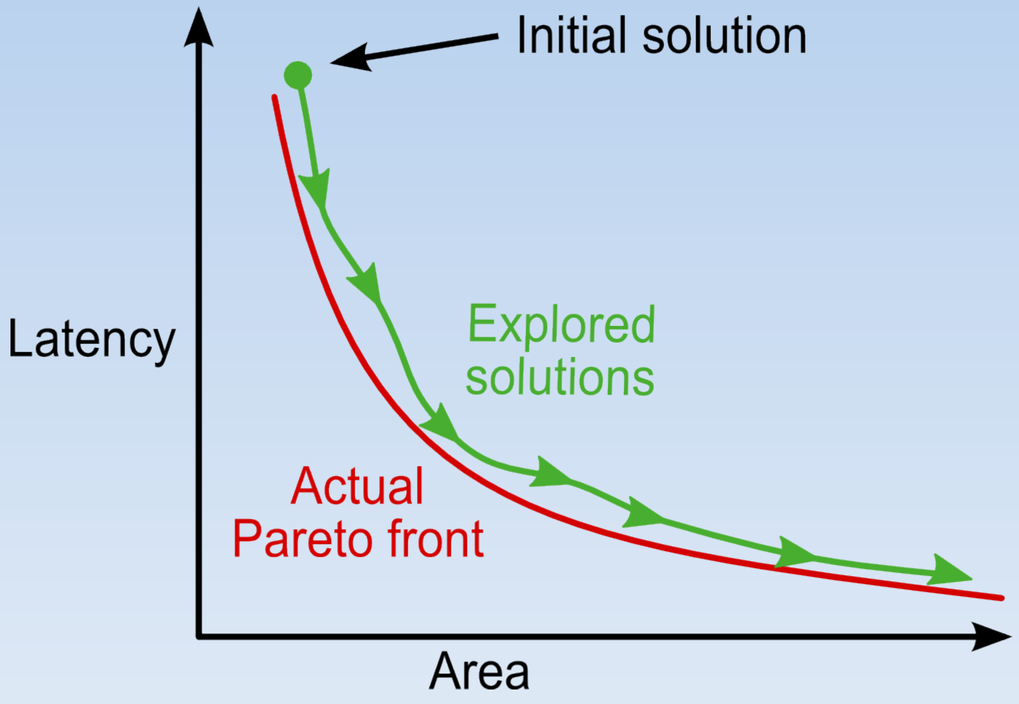
1) Respect contraints

2) Find an efficient solution

Exploration algorithm:

- Start: small circuit

- Then, iterative transformations



Respecting constraints

Exploration priorities:

1) Respect contraints

2) Find an efficient solution

Exploration algorithm:

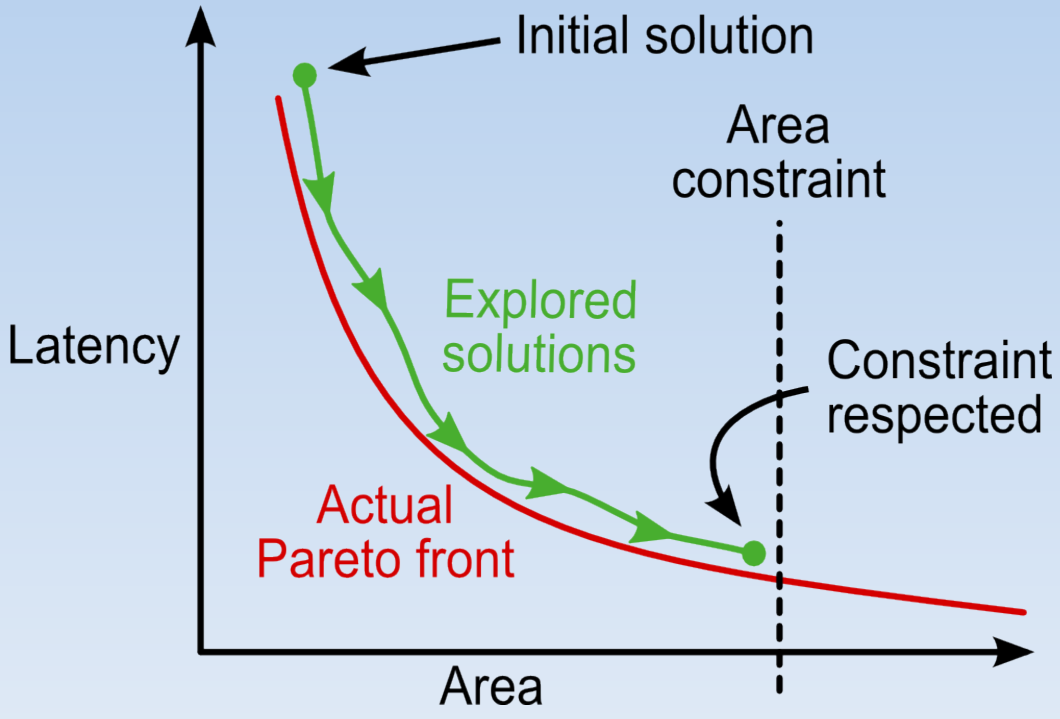
- Start: small circuit

- Then, iterative transformations

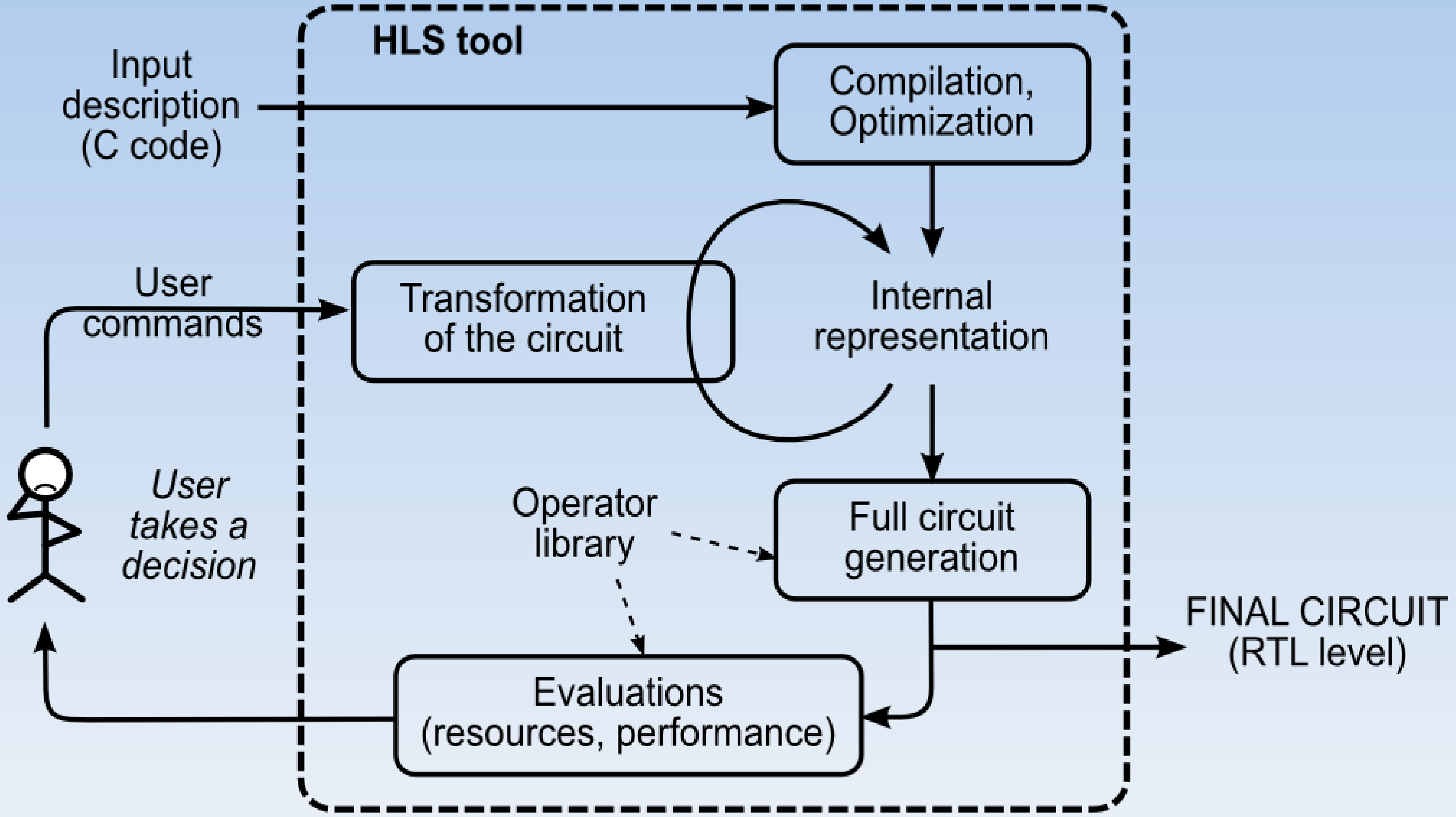
Convergence:

- Greedy progression

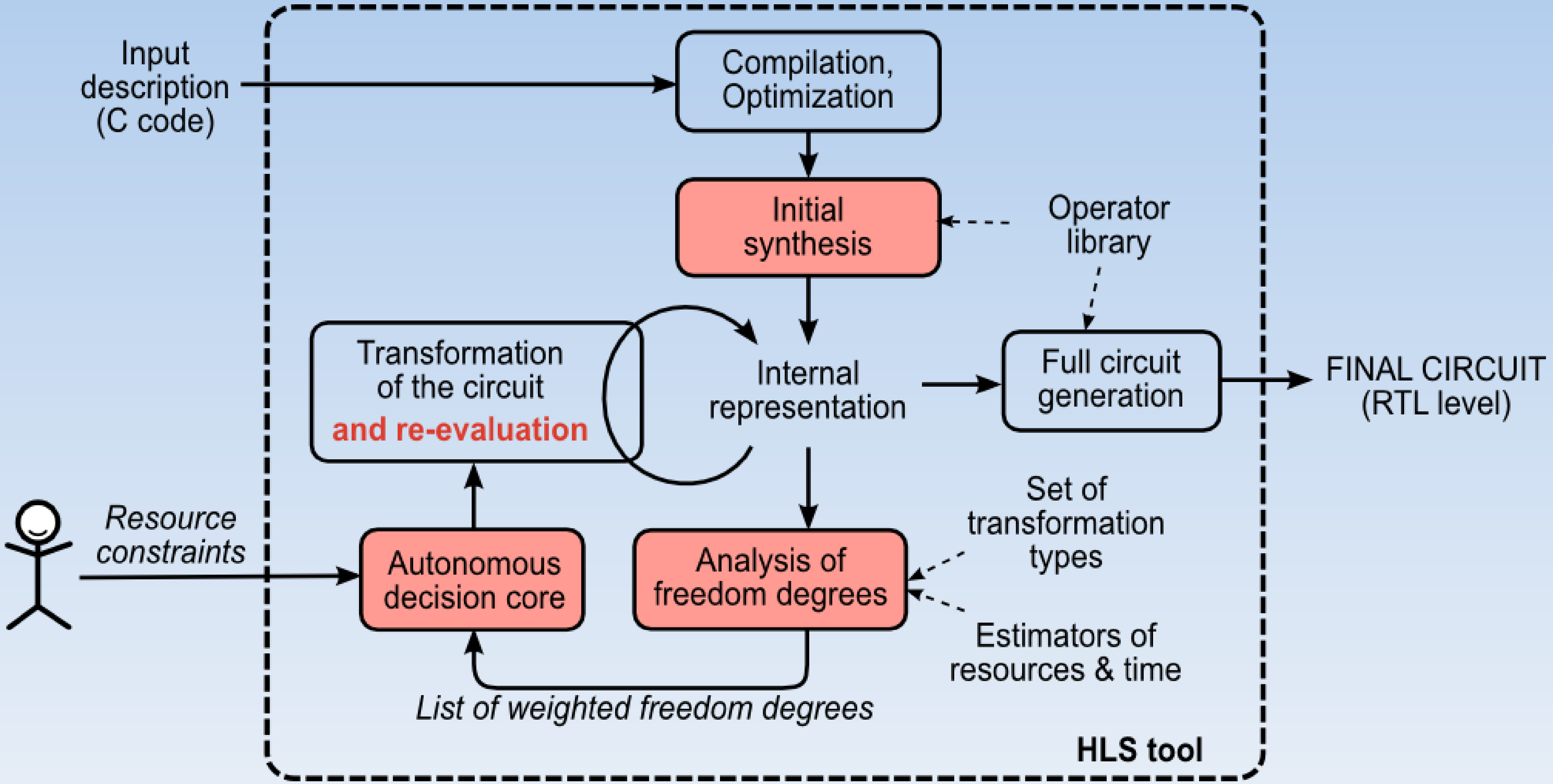
- Guaranteed convergence



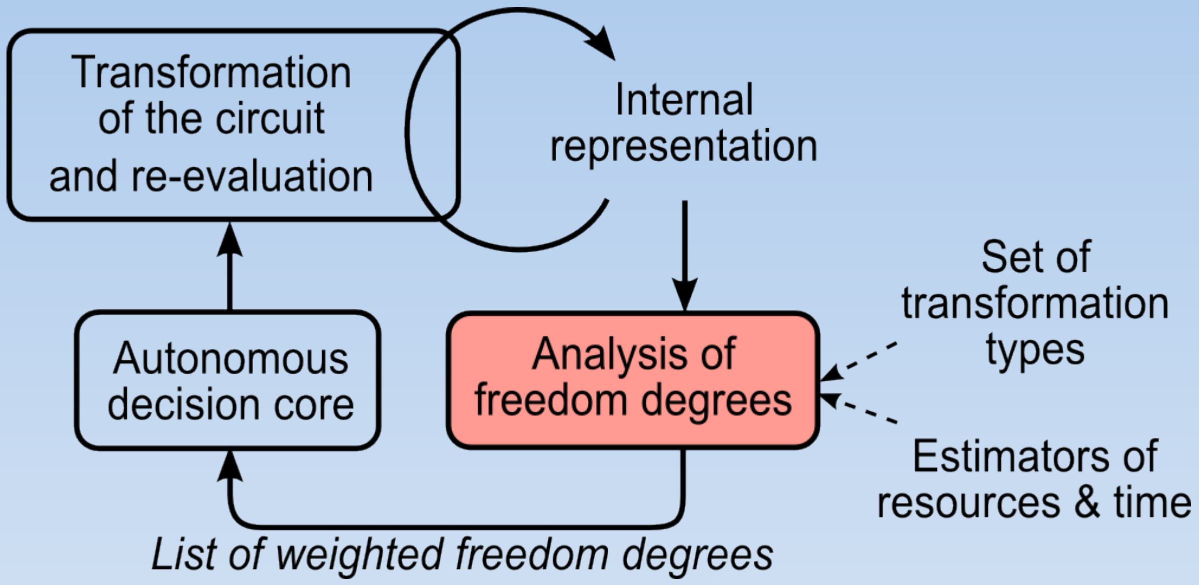
Structure of current HLS tools


HLS tool
Input description (C code)
Compilation, Optimization
User commands
Transformation of the circuit
Internal representation
User takes a decision
Operator library
Full circuit generation
FINAL CIRCUIT (RTL level)
Evaluations (resources, performance)

Proposed HLS tool structure


Input description (C code)
Compilation, Optimization
Initial synthesis
Operator library
Transformation of the circuit
and re-evaluation
Internal representation
Full circuit generation
FINAL CIRCUIT (RTL level)
Resource constraints
Autonomous decision core
Analysis of freedom degrees
Set of transformation types
Estimators of resources & time
List of weighted freedom degrees
HLS tool

Analysis of freedom degrees


Transformation of the circuit and re-evaluation
Internal representation
Set of transformation types
Autonomous decision core
Analysis of freedom degrees
Estimators of resources & time
List of weighted freedom degrees

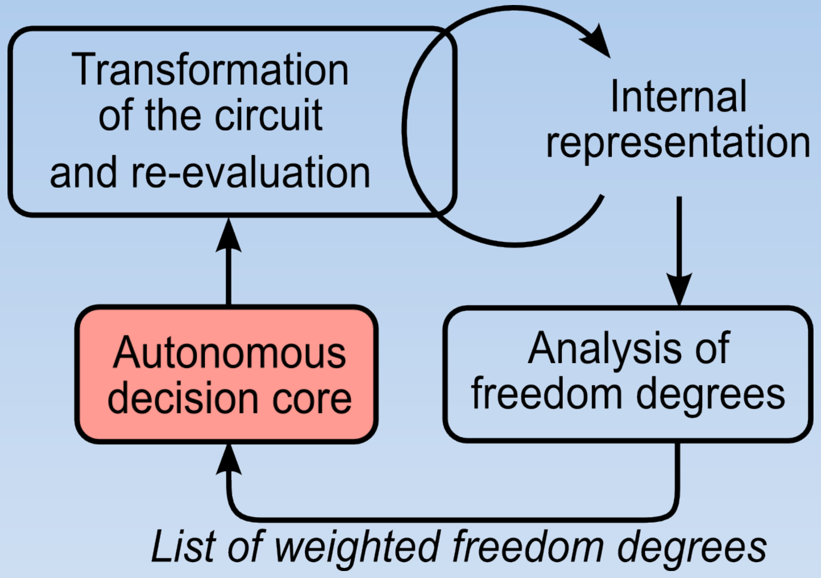
Transformation types:
- Loop unrolling
- Adding operators
...

***Freedom degree* =**
One transformation applied to one place of the circuit

Weighting (estimations):
- Resource cost
- Circuit acceleration

Estimators:
- Forecast of the consequences of a freedom degree
- Dedicated to a transformation type

Choice of the most appropriate freedom degrees


Transformation of the circuit and re-evaluation
Internal representation
Autonomous decision core
Analysis of freedom degrees
List of weighted freedom degrees

Final weighting of freedom degrees: a measure of the ”global” quality

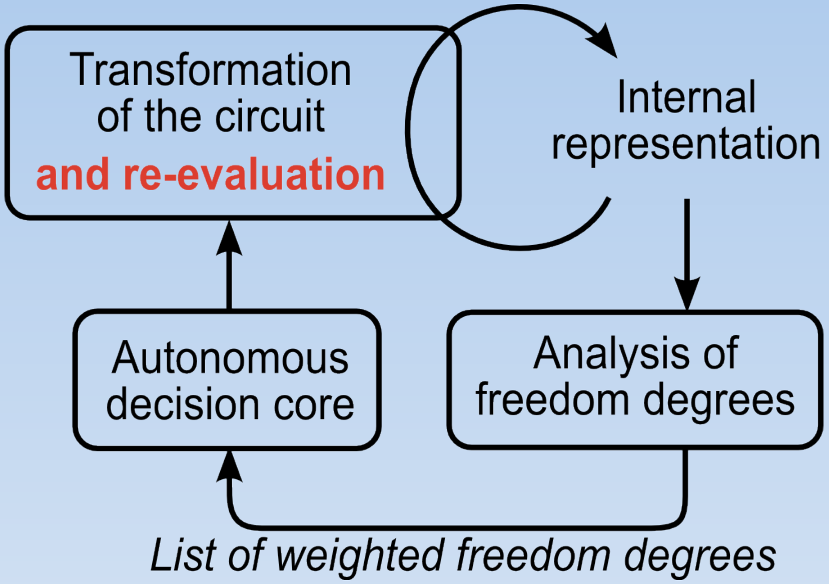
Computed from:
- Resources cost (estimated)
- Circuit acceleration (estimated)

$$\textbf{Quality} = \frac{\textbf{Circuit acceleration}}{\textbf{Resource cost}}$$

Final choice : the freedom degree with best ”quality”.

For DSE rapidity: select extra freedom degrees (within 10% of remaining resources)

Respect contraints: re-evaluation of the solution


Transformation of the circuit and re-evaluation
Internal representation
Autonomous decision core
Analysis of freedom degrees
List of weighted freedom degrees

Objectives of the re-evaluation:

- Verify accuracy of estimations (area)

- Update the internal representation

Provide precise evaluations:

- Generate structural RTL (low optimization)

- Use operator library characterizations

- Take all entities into account (MUX, FSM, etc)

Host HLS tools

Choosing the host HLS tool

- **Availability of the source code**
- **Precise operator library**
- **Allocation-first strategy**
- **Complexity level: accessible**

The UGH tool

- **UGH = User Guided High-level synthesis**
- **Open-source, academic tool**
- **Developped at LIP6 (Paris) & TIMA (Grenoble)**
- **Simple internal generation flow**

Modification de UGH

Main modifications

- **Operator library:**
  Calibration fo technologies Xilinx Virtex-5
- **Extend internal representation: hierarchy**
- **Replaced scheduler, mapper and retiming**
- **Added exploration core**

Source code

- **UGH: ~450k lines of C/C++**
- **Added ~36k lines of C code, much removed...**
- **AUGH: ~150k lines of C code**

Transformation types

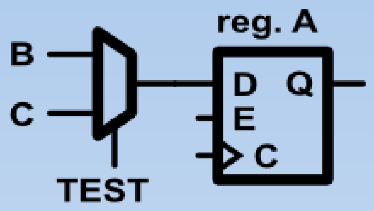
- **Adding operators**
- **Loop unrolling**
- **Condition wiring**

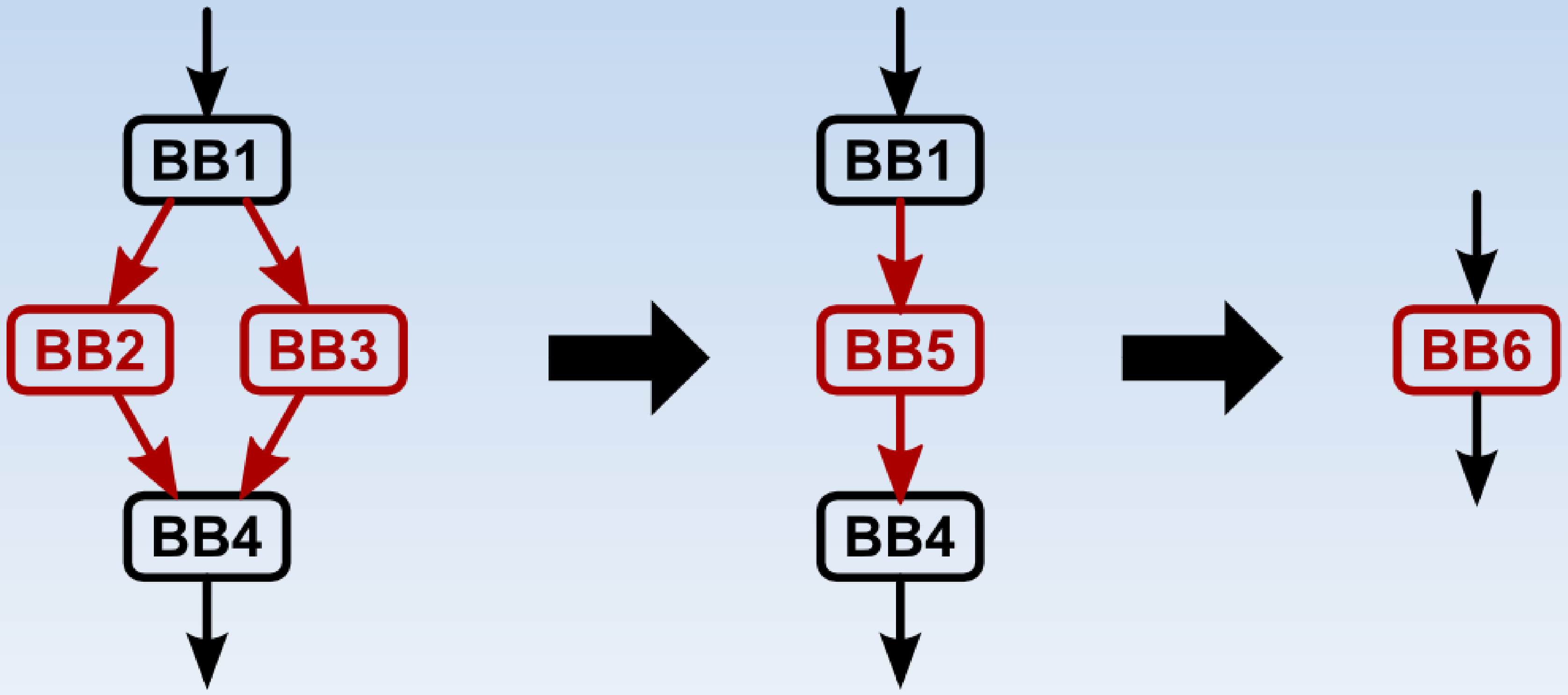
Condition wiring

```
if(TEST) A = B;
else     A = C;
```

→

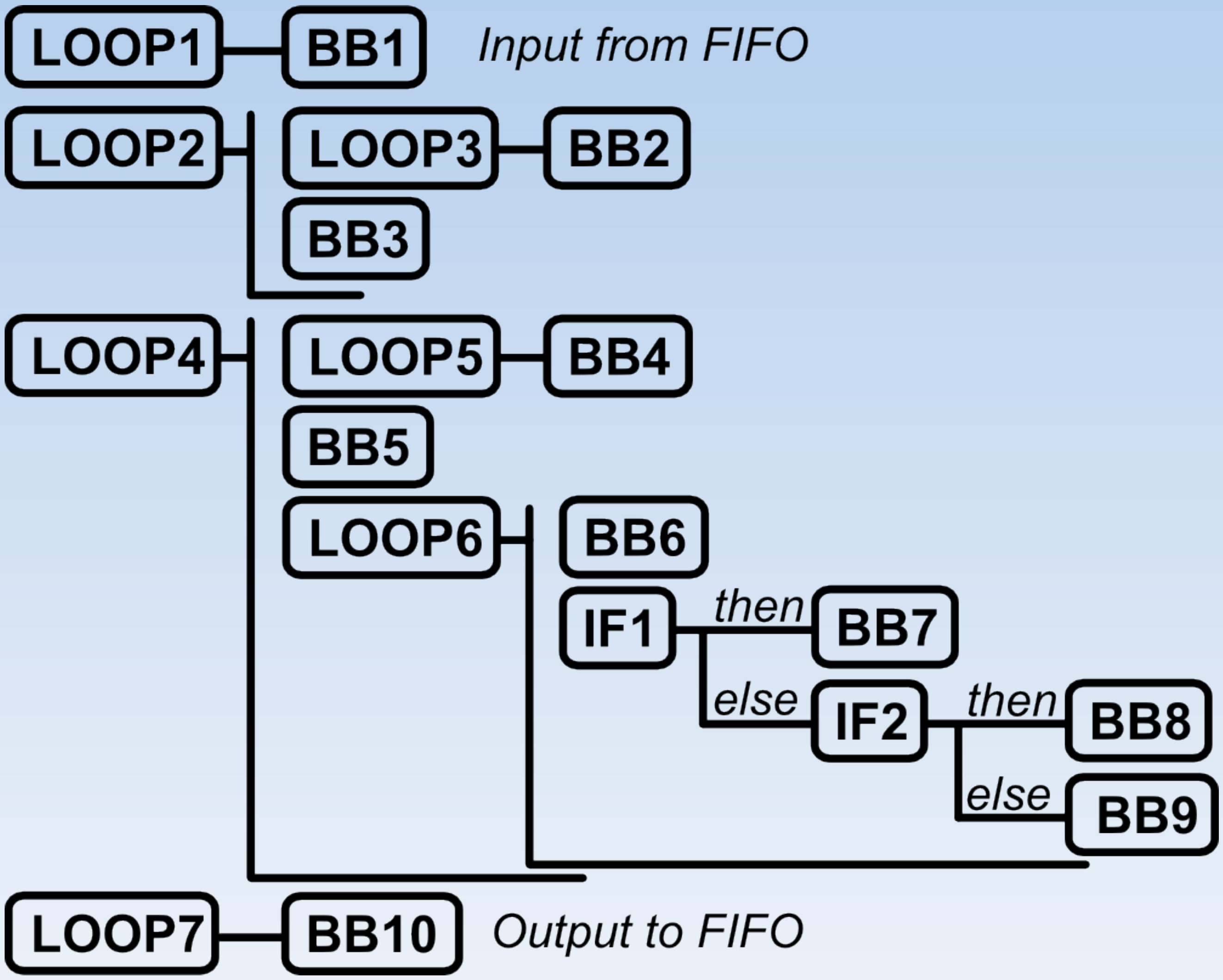
```
A = TEST ? B : C;
```



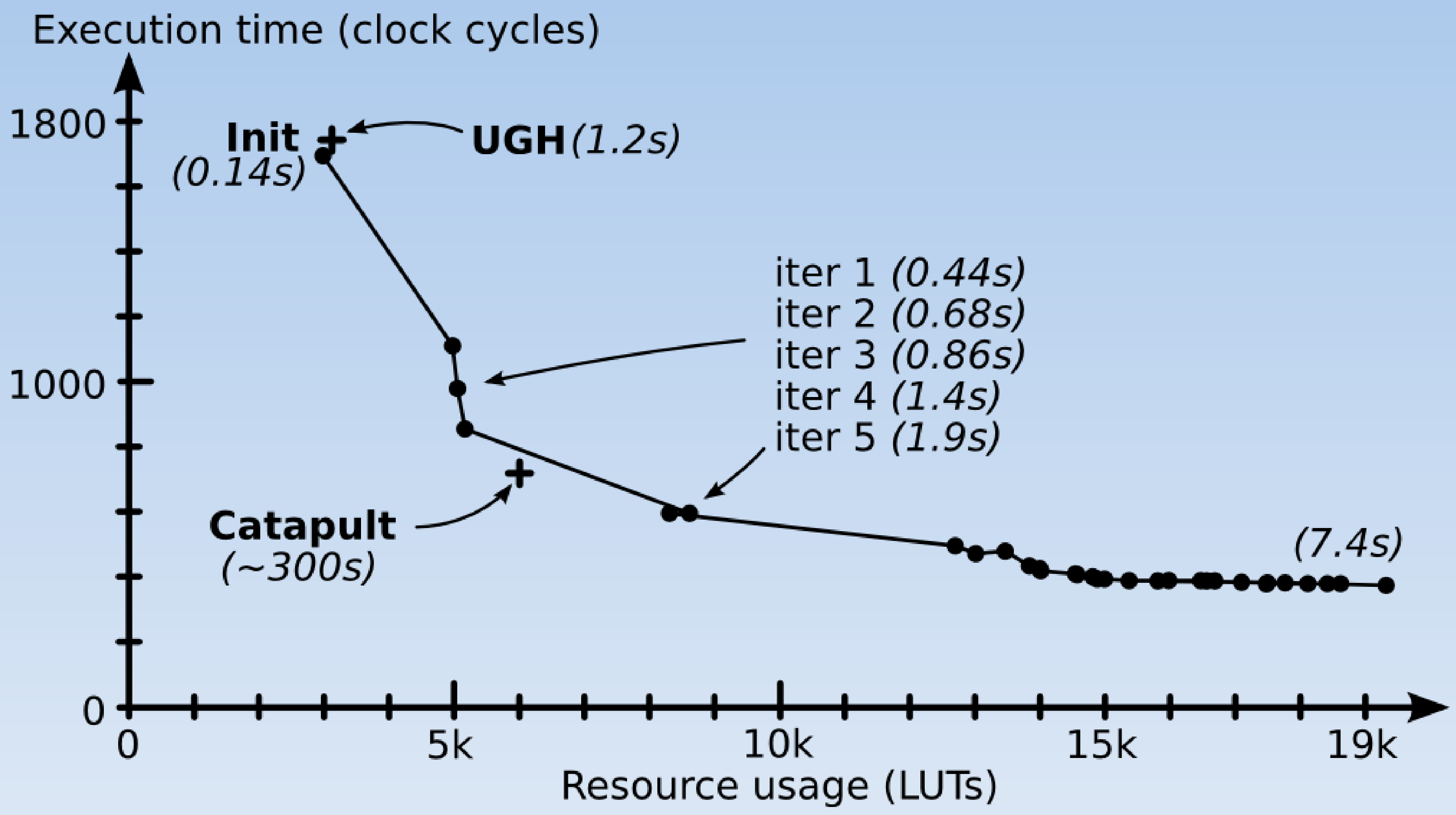


Synthesis of the 2D IDCT 8x8

Structure of the algorithm IDCT 2D (Loeffler implem.)



Results



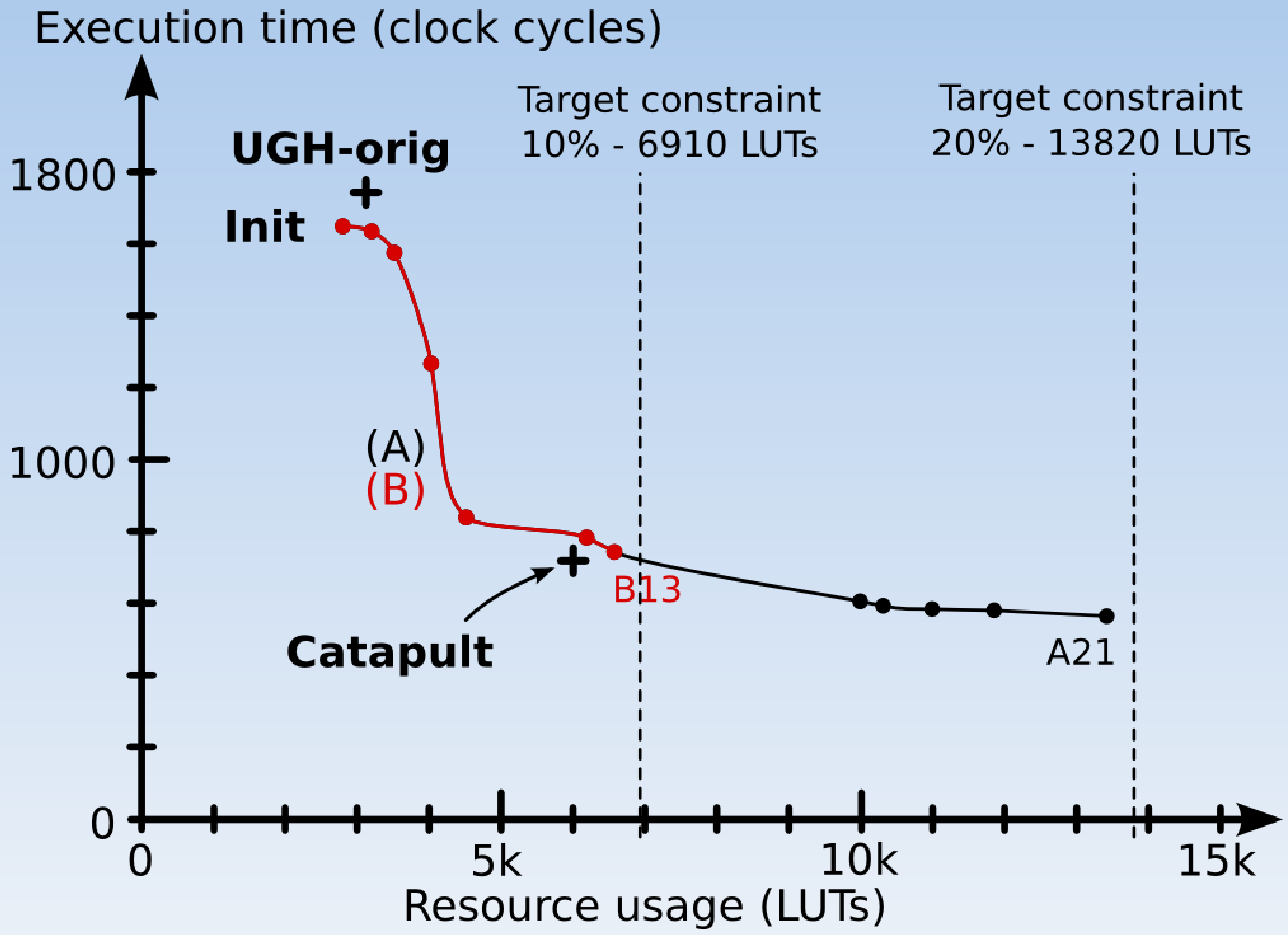
Experiment UGH:
duration 1.2s

Experiment Catapult:
duration 300s

Experiment Vivado HLS:
duration 60s

Experiment AUGH:
duration 7.4s
iterations: 20

Respect of resource constraints (older results)



Conclusion

We have proposed a new HLS methodology:

- **DSE is autonomous**
- **DSE is fast**
- **Resource constraints are respected**
- **Flow close to compilation**

Design for FPGA more accessible to non-experts.

Perspectives

Further works being done:

- **Additional transformation types**
- **Multi-port memories**
- **Scheduler improvements**
- **Calibrations for other technologies**

Important contribution to the open-source community