UNIVERSITEIT GENT

FACULTEIT INGENIEURSWETENSCHAPPEN

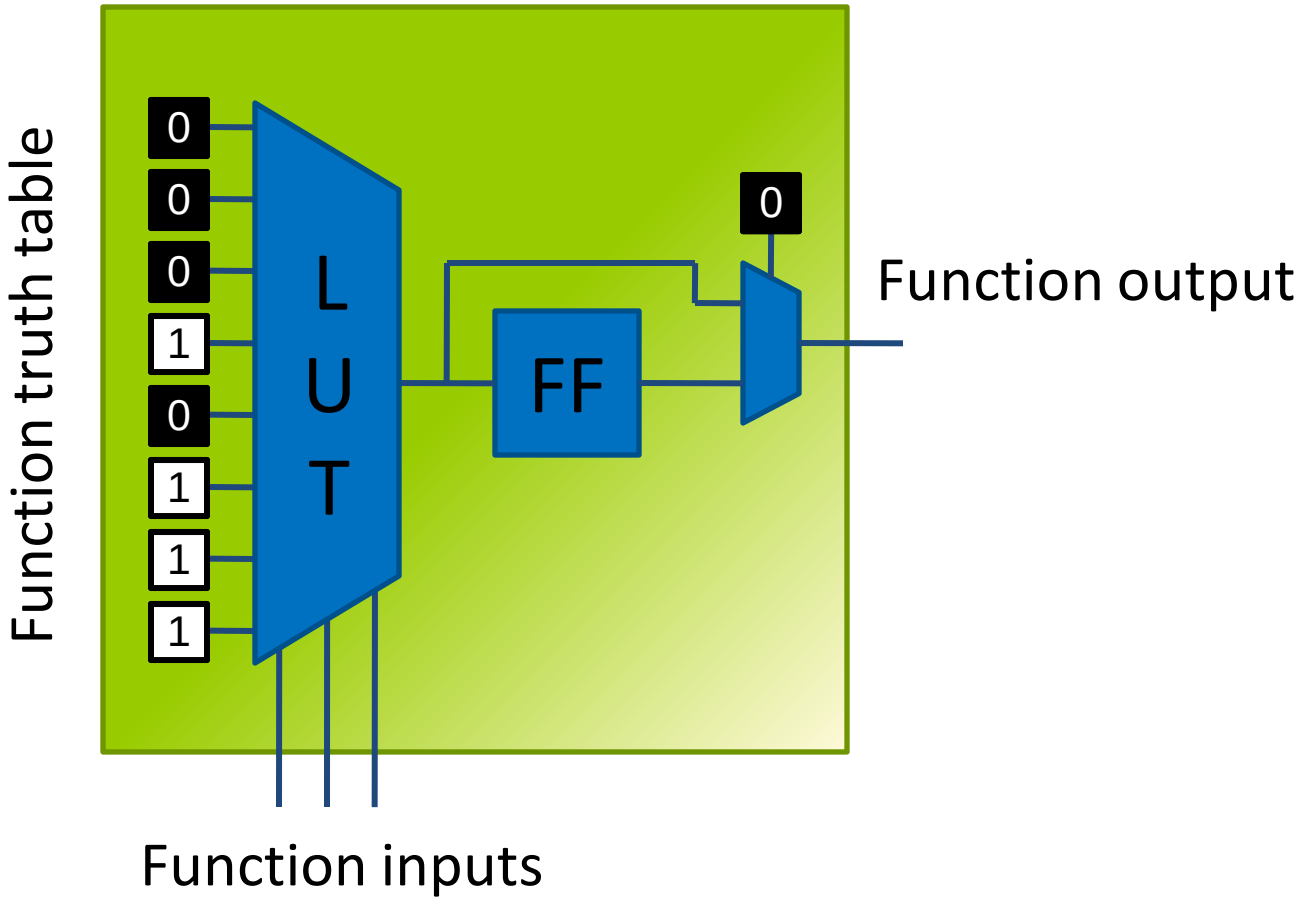
Supersnelle runtime-herconfiguratie eindelijk binnen handbereik

Prof. Dirk Stroobandt, Universiteit Gent

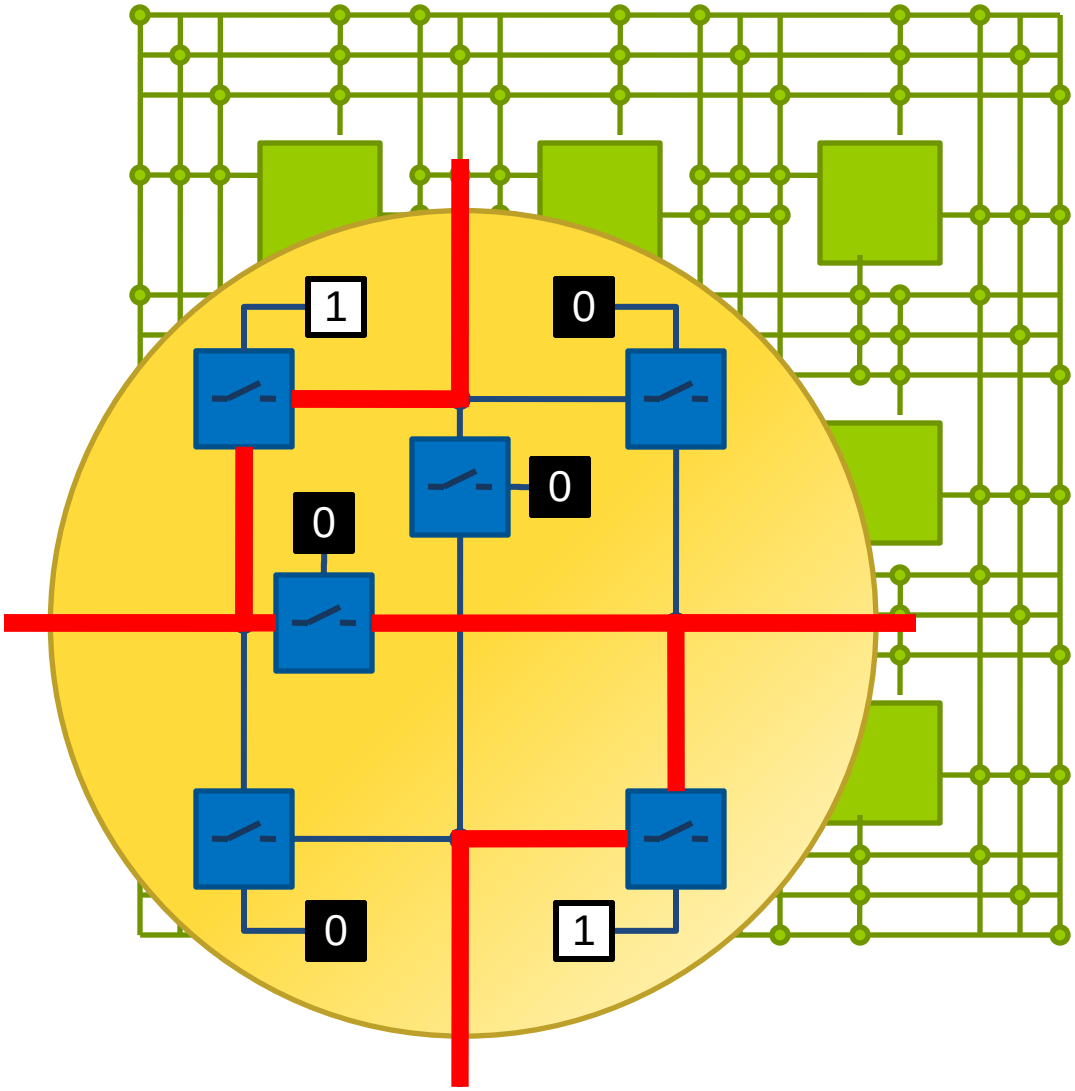
Onderzoekswerk uitgevoerd door

Karel Bruneel

FPGAs are configurable: logic

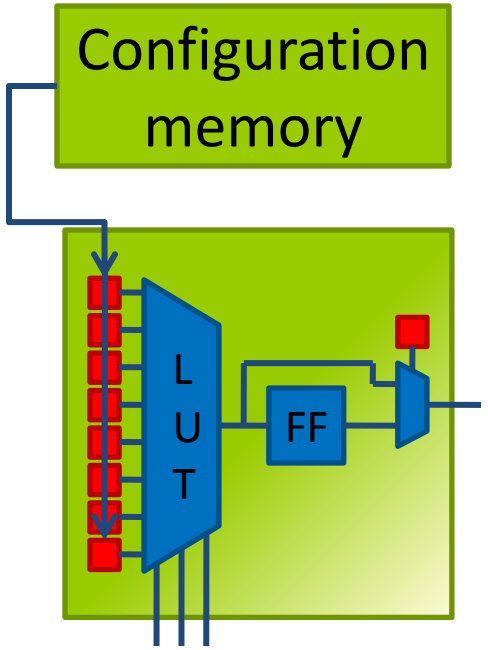


FPGAs are configurable: interconnect



FPGAs are reconfigurable!

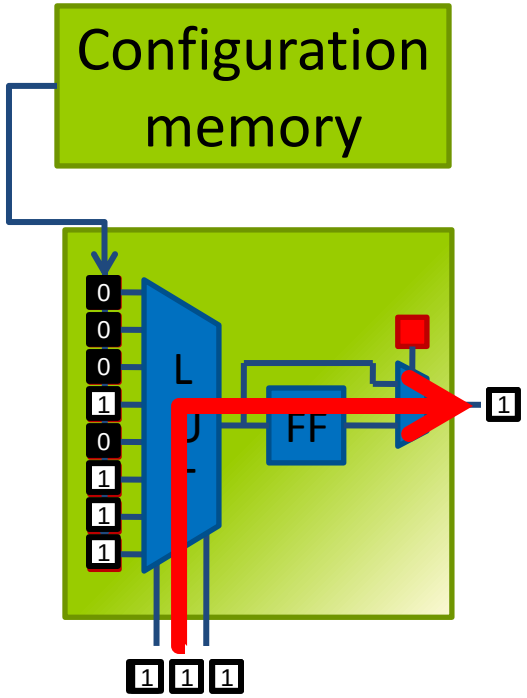
- FPGA's functionality determined by content configuration memory
- Configuration memory easily rewritten = reconfigured



FPGA's functionality can be changed at run-time

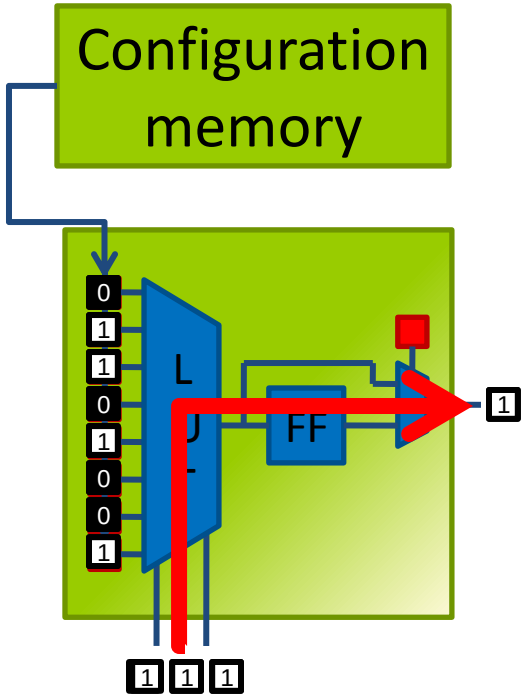
Using reconfigurability

- Today: configurability on a ***large time scale***
  - Only when the system starts
  - Prototypes
  - …



Using reconfigurability

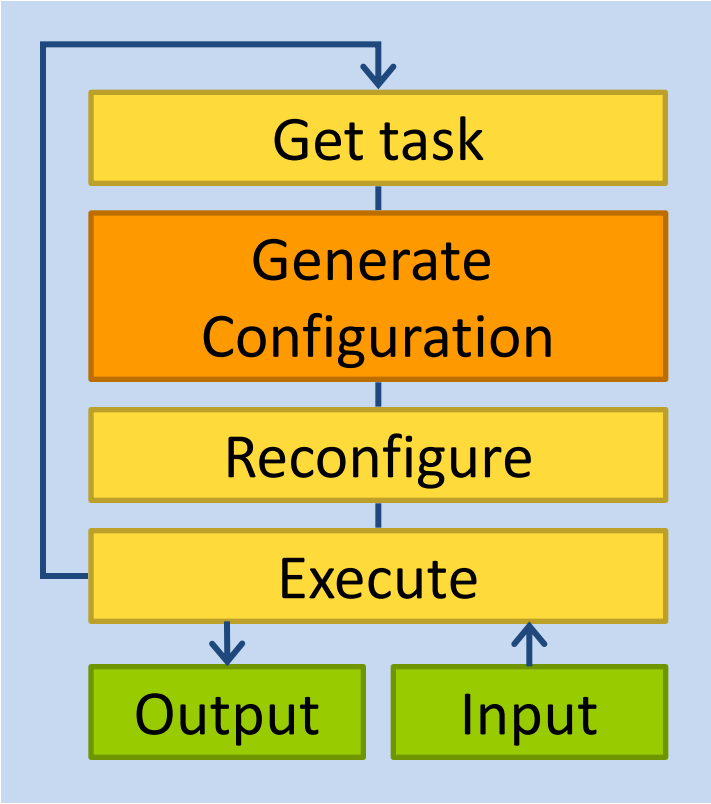
- Research community: configurability on a ***smaller time scale***
  - run-time reconfiguration
  - Goals:
    - Improve performance
    - Reduce area
    - Reduce power consumption



Run-time reconfiguration

Functionality optimized for the exact problem at hand

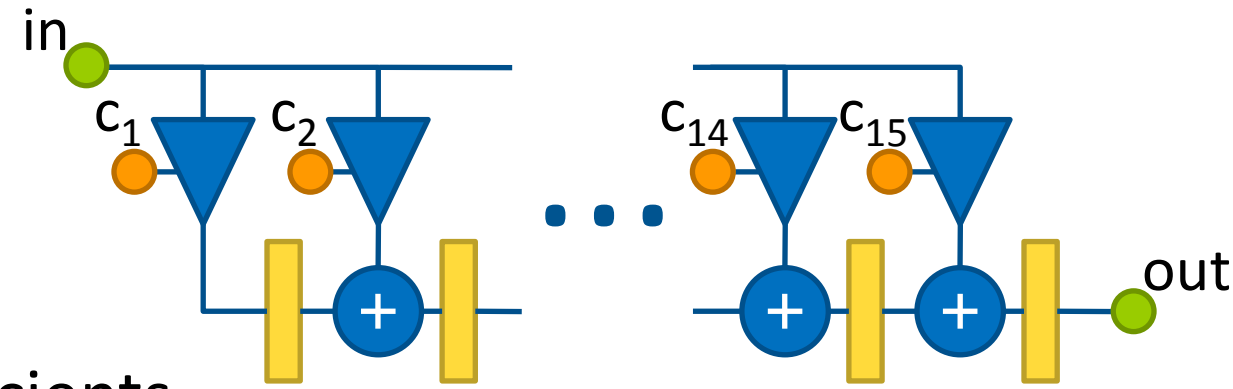
On-line



- At run-time more details about the task are known
- Leads to
  - smaller hardware
  - faster hardware
  - more power efficiency

Example: adaptive filtering

- FIR filter
  - 16 taps
  - 8-bit input
  - 8-bit coefficients



- Now used: static implementation
  - Coefficients are treated as inputs
  - Most flexible implementation
  - Large and slow

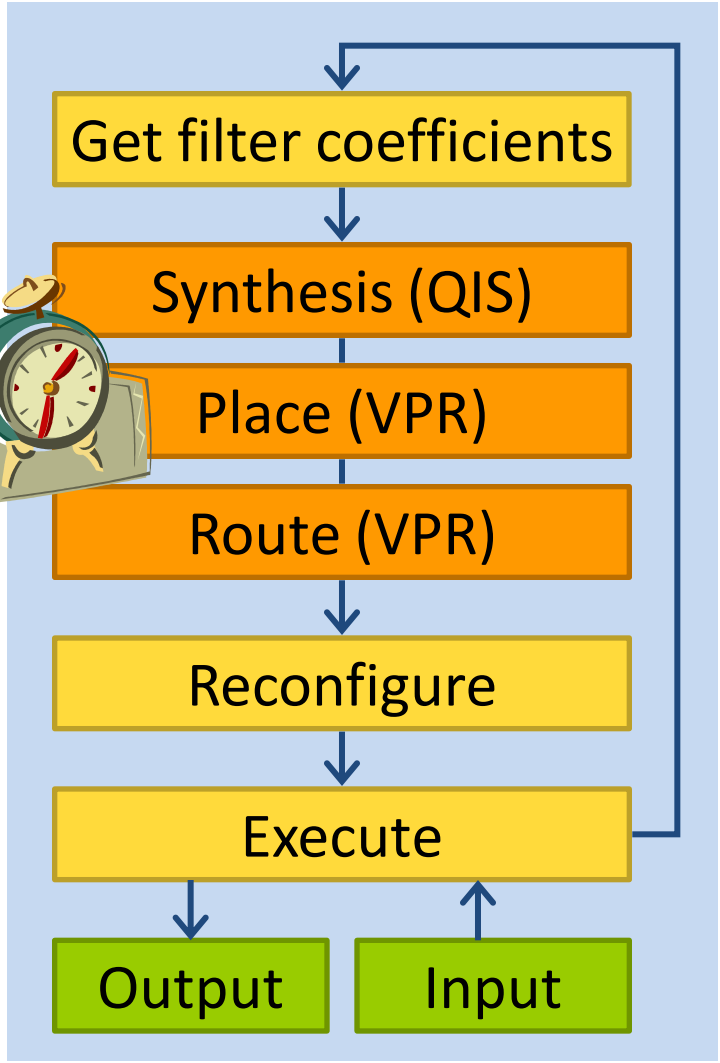
Static

2999 LBs

8.4 MHz

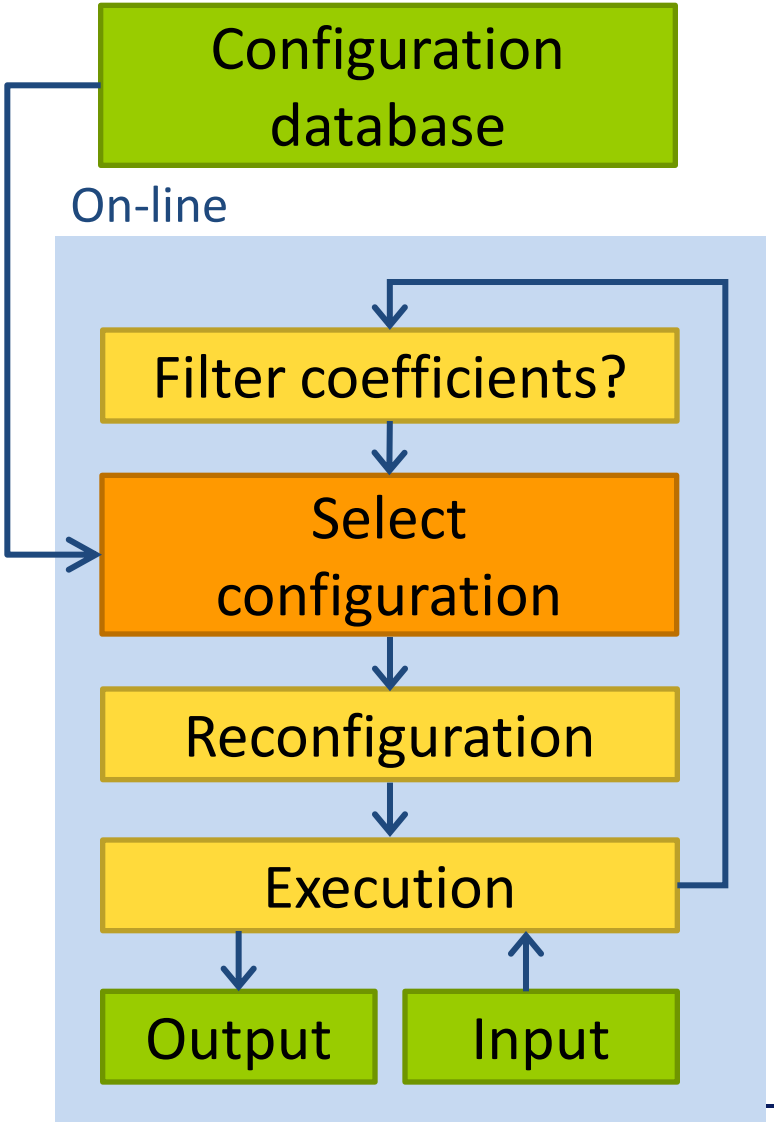
Conventional runtime reconfiguration

On-line



- Static ↔ Dynamic
  - 2999 LBs ↔ 1005 LBs
  - 66% smaller
  - 8.4 MHz ↔ 11.9 MHz
  - 29% faster
- Generation overhead
  - 35 seconds!

Off-line generation is impossible!



- Off-line ↔ On-line

  2999 LBs ↔ 1005 LBs

  66% smaller

  8.4 MHz ↔ 11.9 MHz

  29% faster

- Database size!

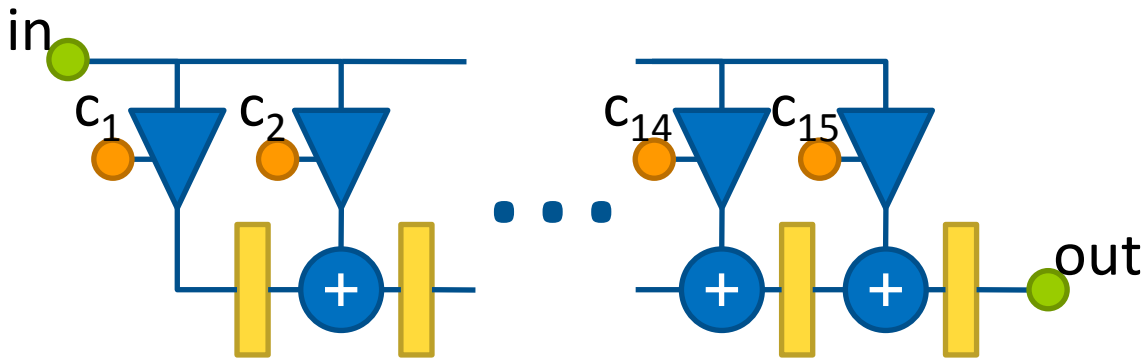
$$2^{16\times 8} = 2^{128}$$

Introduction summary

- FPGAs are reconfigurable
- Run-time reconfiguration can:
  - Improve performance
  - Reduce area and power
- Problem:
  - Conventional toolflow is too slow
  - Full flexibility requires too much memory

How to avoid full reconfiguration?

- A lot of problems have parameters
  - Change a lot less frequently than general inputs
  - Define a specific implementation
  - Allow optimized implementations for each value
- FIR-filter example
  - Filter coefficients = parameters

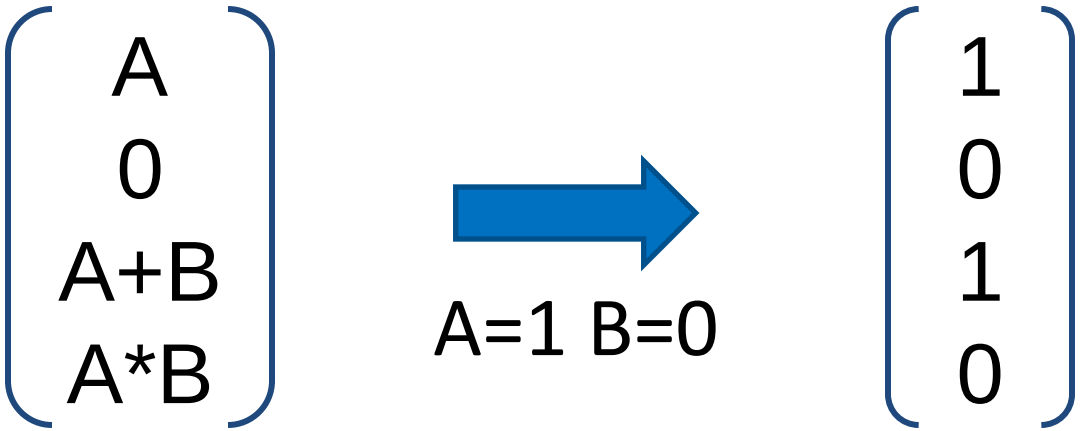


Our solution!

- Move as much work as possible to compile time
  - Without restricting solution space!
  - Hope: good quality HW in shorter time
- At compile time
  - Start with generic problem + parameter set
  - Use information to generate parameterized solution
- At run time
  - Find out what the specific problem is
  - Specialize very fast, reconfigure and execute

How: parameterizable configurations

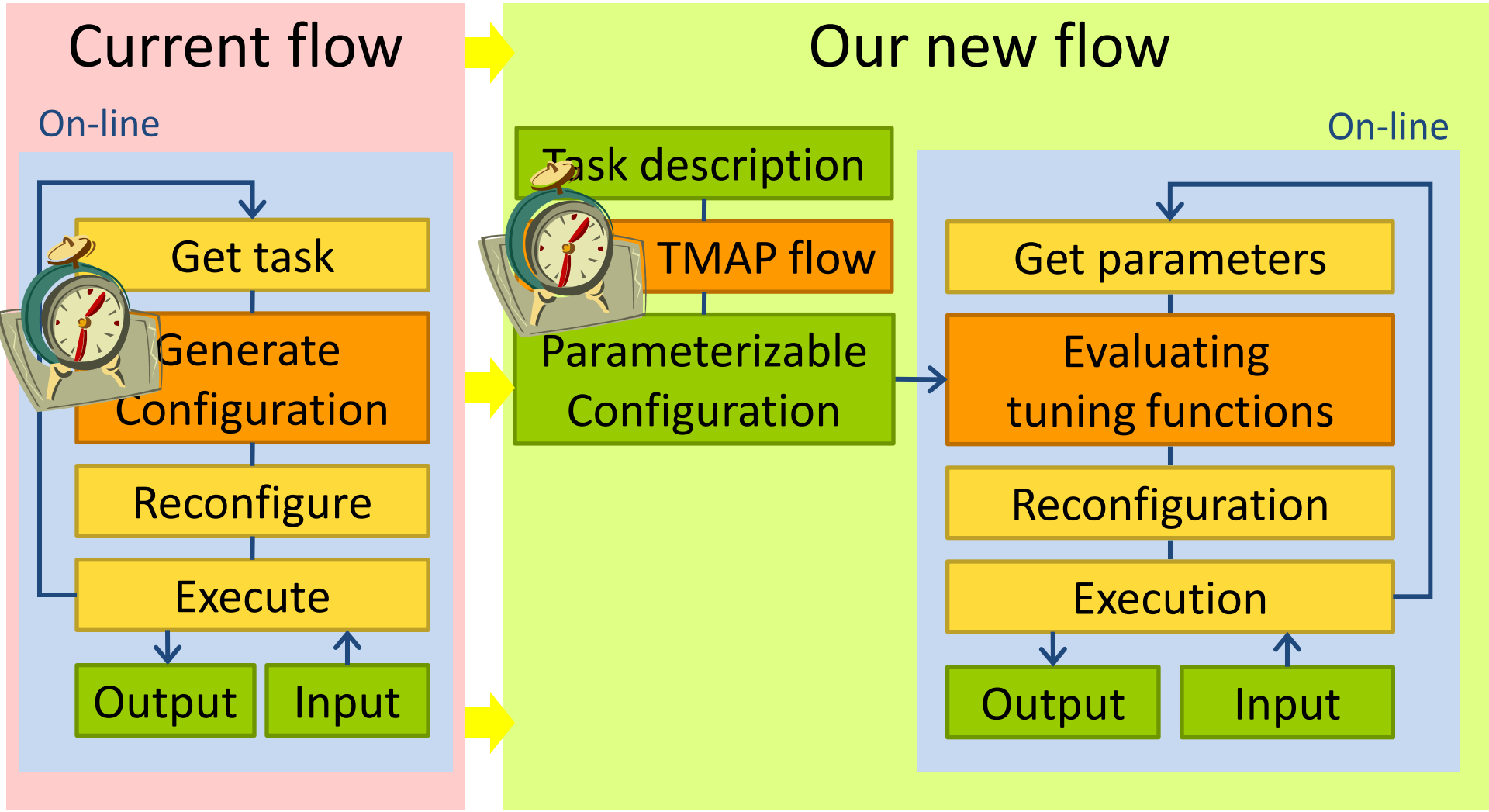
- Regular configuration: vector of 1's and 0's
- Parameterizable configuration:
  - Vector: one element for every configuration bit
  - Elements: boolean functions (tuning functions)
  - Tuning functions depend on a set of parameters



Properties

- Evaluation of boolean functions is fast
  - Much faster than full synthesis flow
- Compact representation
  - A lot more compact than storing all possibilities as vectors

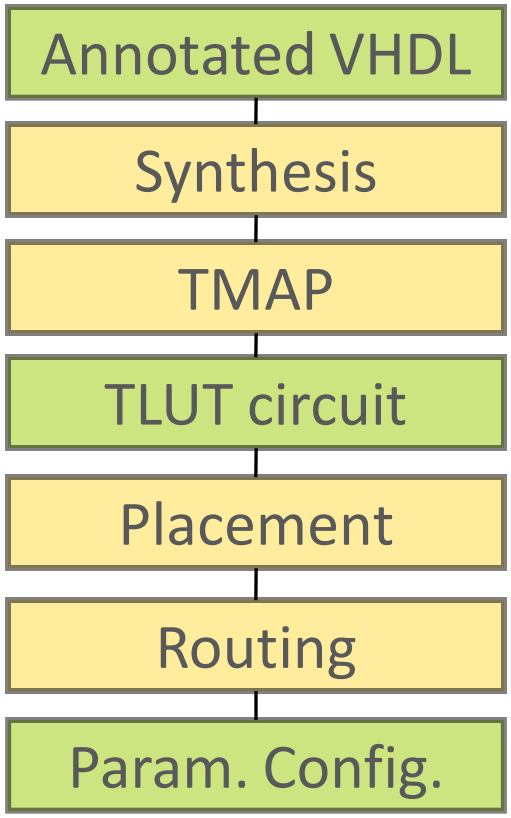
New tool flow



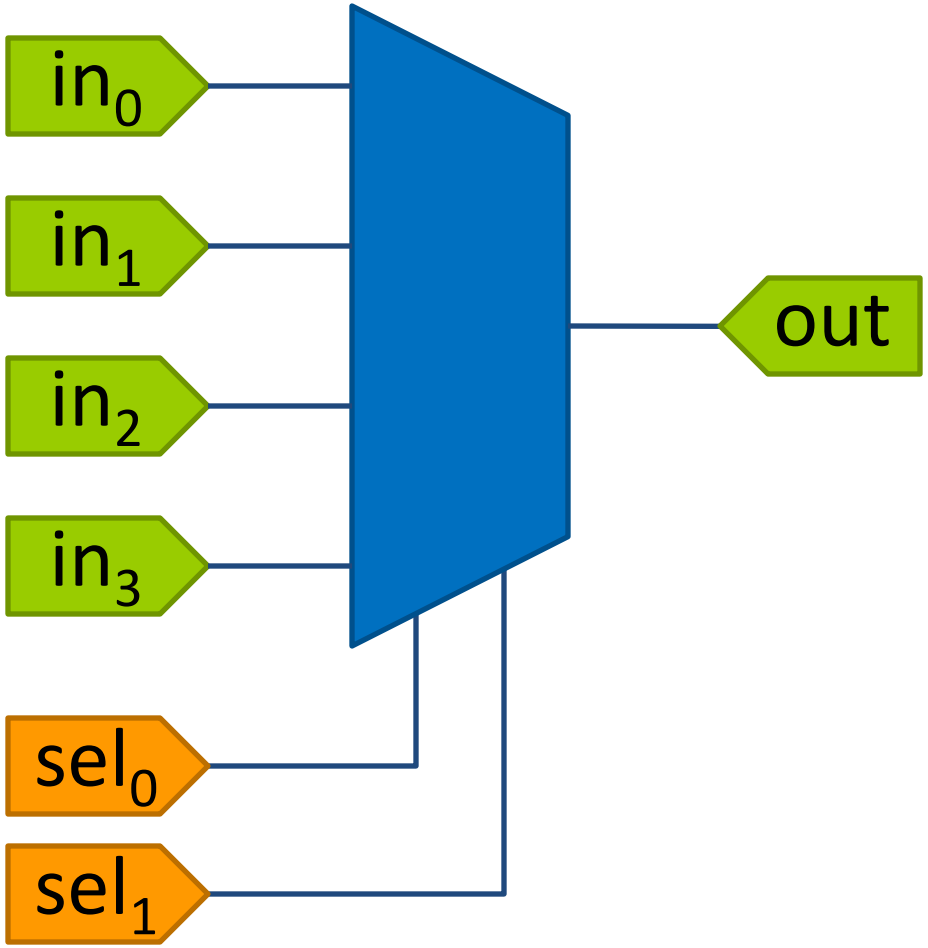
TMAP flow

- Fully automatic toolflow
- Input:
  - A circuit description
  - The parameter inputs (subset of inputs)
- Intermediate: tunable LUT circuit
- Output:
  - A parameterizable configuration
- Only the LUT configuration bits are tunable

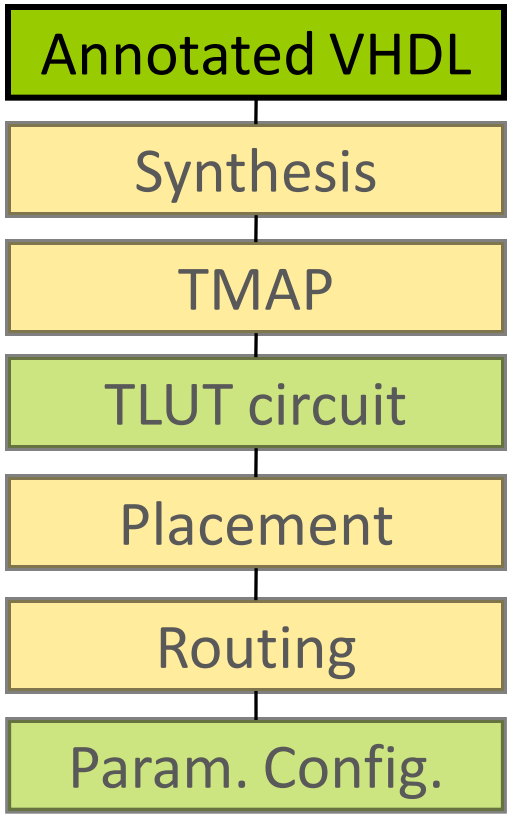
TMAP flow



e.g. 4:1 multiplexer

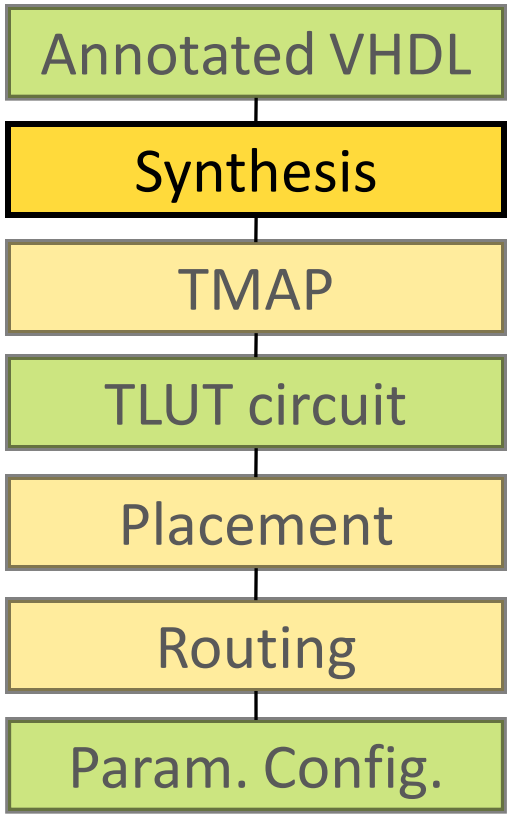


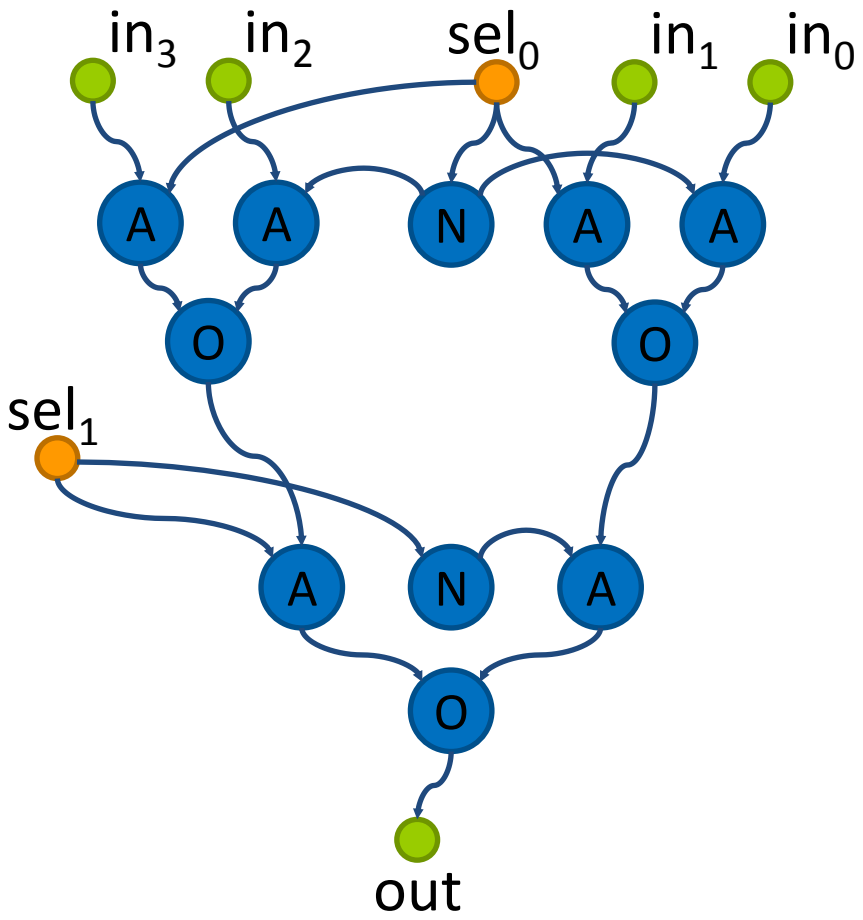
Annotated VHDL



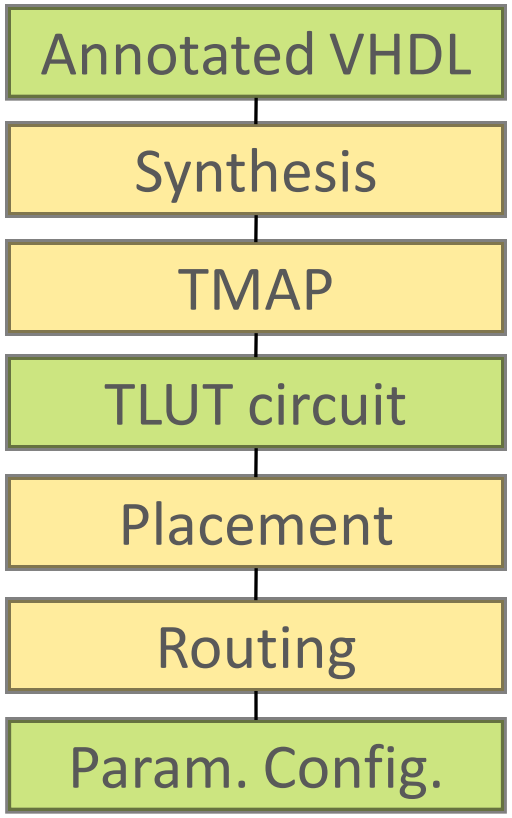

```
entity multiplexer is
port(
  in  : in  std_logic_vector(3 downto 0);
  --PARAM
  sel : in  std_logic_vector(1 downto 0);
  --PARAM
  out : out std_logic
);
end muxltiplexer;

architecture behavior of multiplexer is
begin
  out <= in(conv_integer(sel));
end behavior;
```

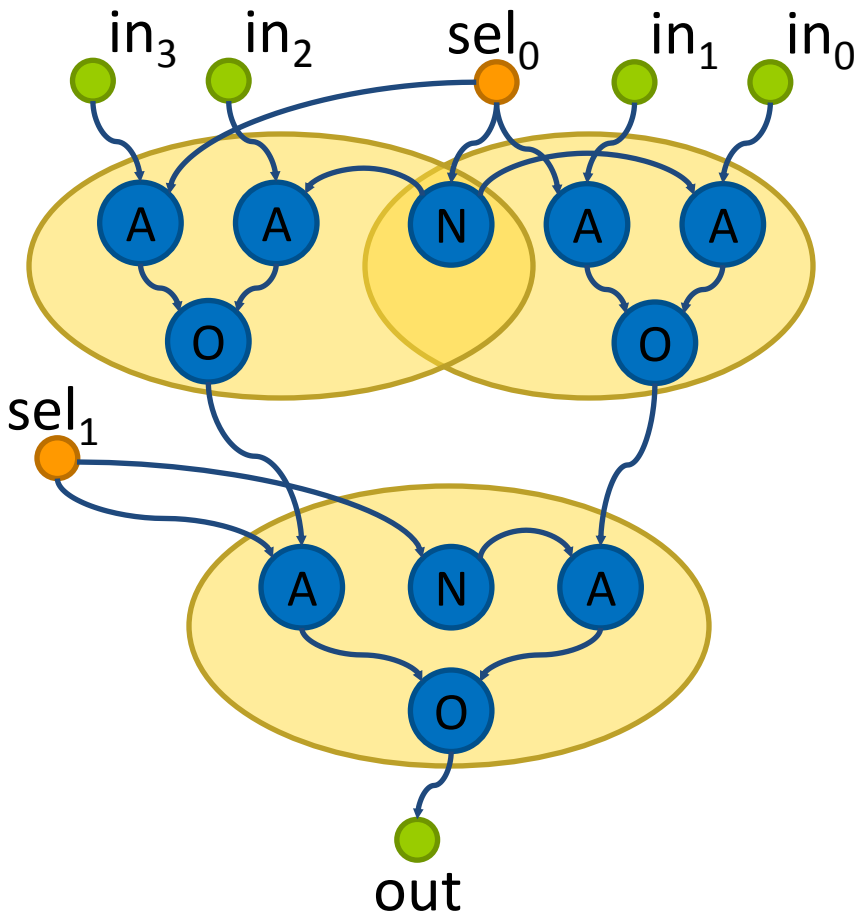
Synthesis


Annotated VHDL
Synthesis
TMAP
TLUT circuit
Placement
Routing
Param. Config.


in3
in2
sel0
in1
in0
A
A
N
A
A
O
O
sel1
A
N
A
O
out

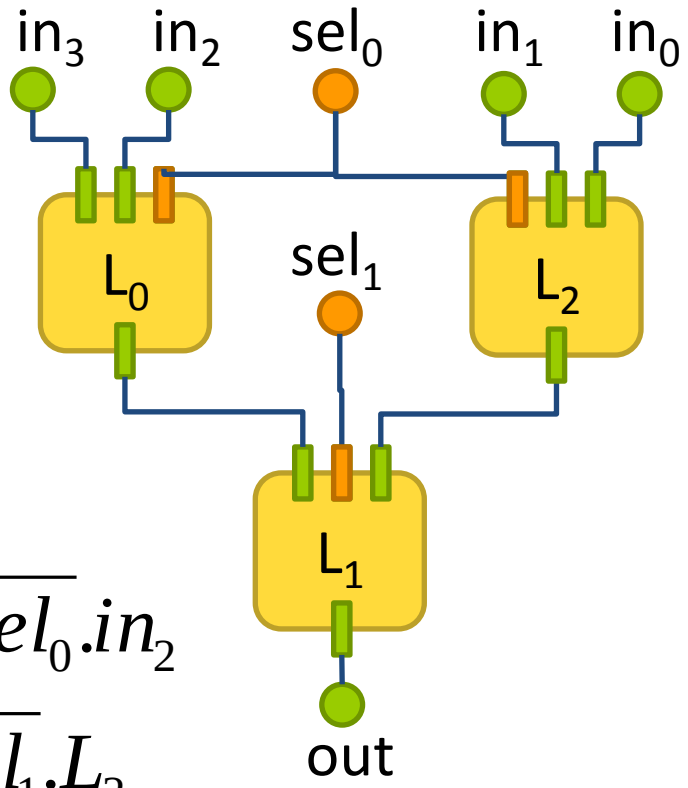
Conventional technology mapping





Conventional LUT circuit

- Annotated VHDL
- Synthesis
- TMAP
- TLUT circuit
- Placement
- Routing
- Param. Config.

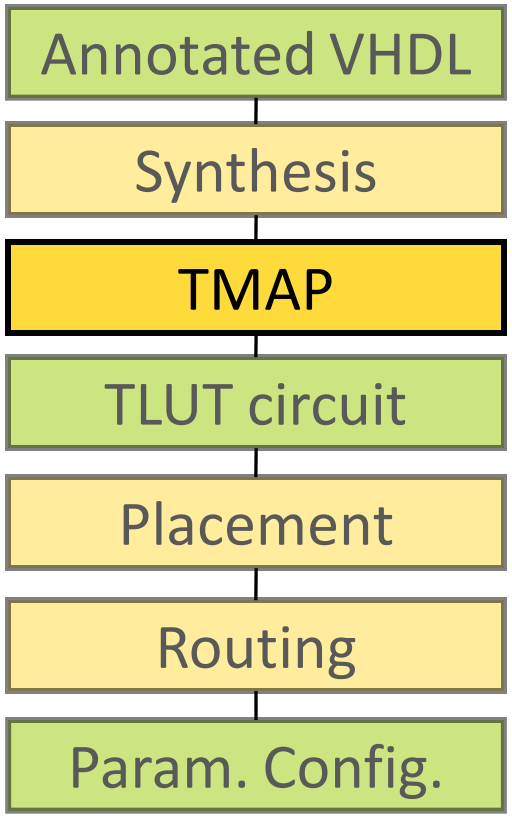


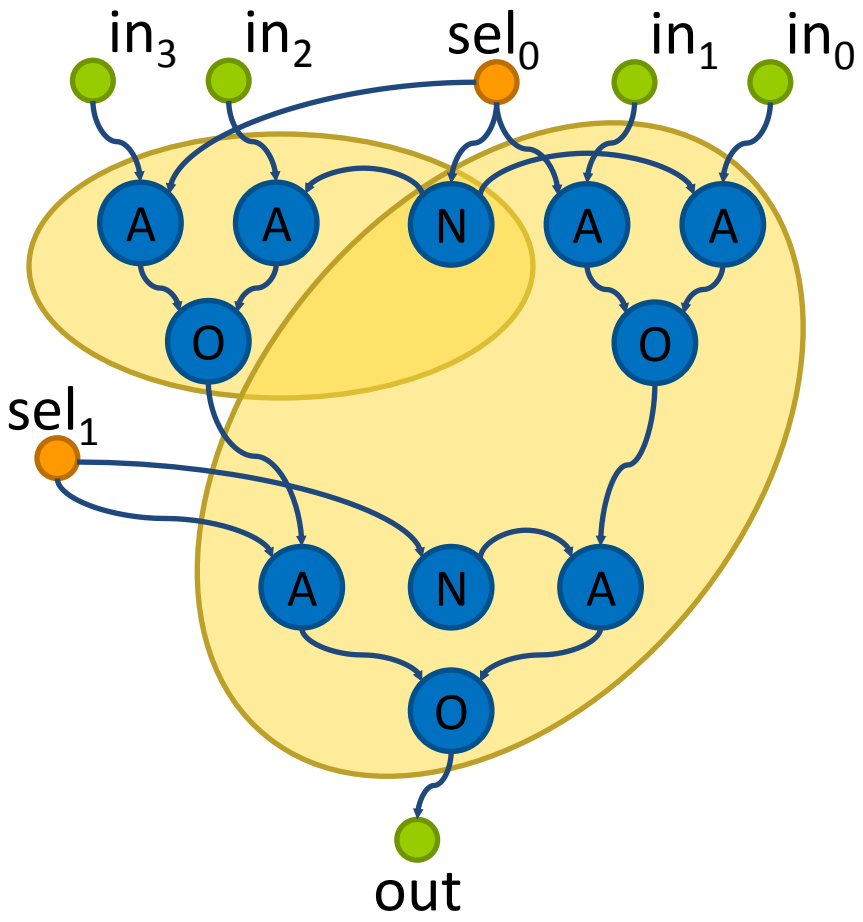
$$L_0 = sel_0.in_3 + \overline{sel_0}.in_2$$

$$L_1 = sel_1.L_0 + \overline{sel_1}.L_2$$

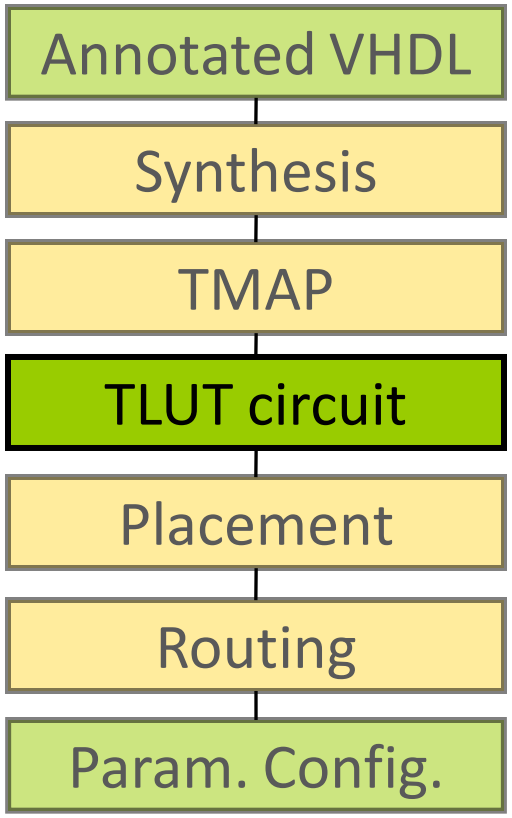
$$L_2 = sel_0.in_1 + \overline{sel_0}.in_0$$

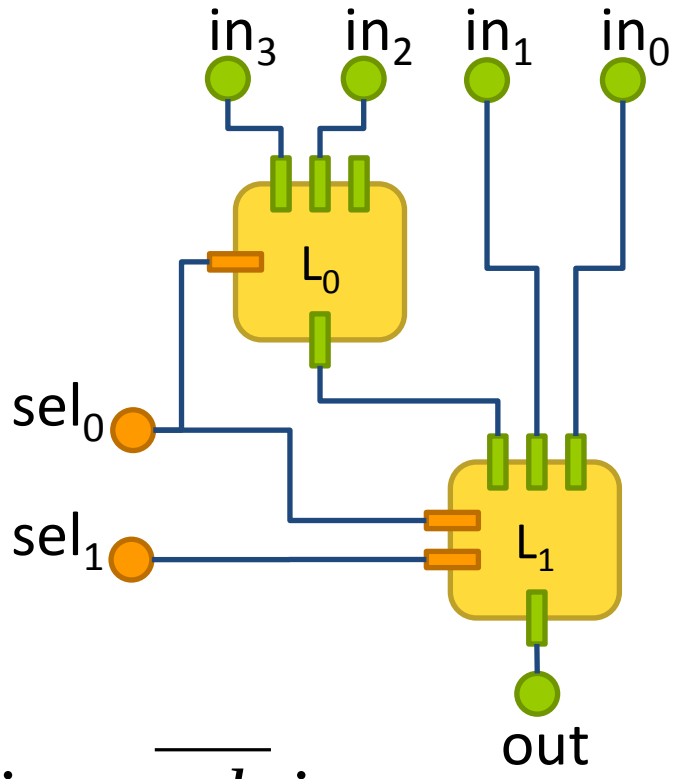
TMAP: Tunable LUT mapping





Tunable LUT circuit





$$L_0 = sel_0.in_3 + \overline{sel_0}.in_2$$

$$L_1 = sel_1.L_0 + \overline{sel_1}.(sel_0.in_1 + \overline{sel_0}.in_0)$$

Extracting tuning functions

$$L_1 = sel_1.L_0 + \overline{sel_1}.(sel_0.in_1 + \overline{sel_0}.in_0)$$

- Annotated VHDL
- Synthesis
- TMAP
- **TLUT circuit**
- Placement
- Routing
- Param. Config.

0

$\overline{sel_1}.\overline{sel_0}$

$\overline{sel_1}.sel_0$

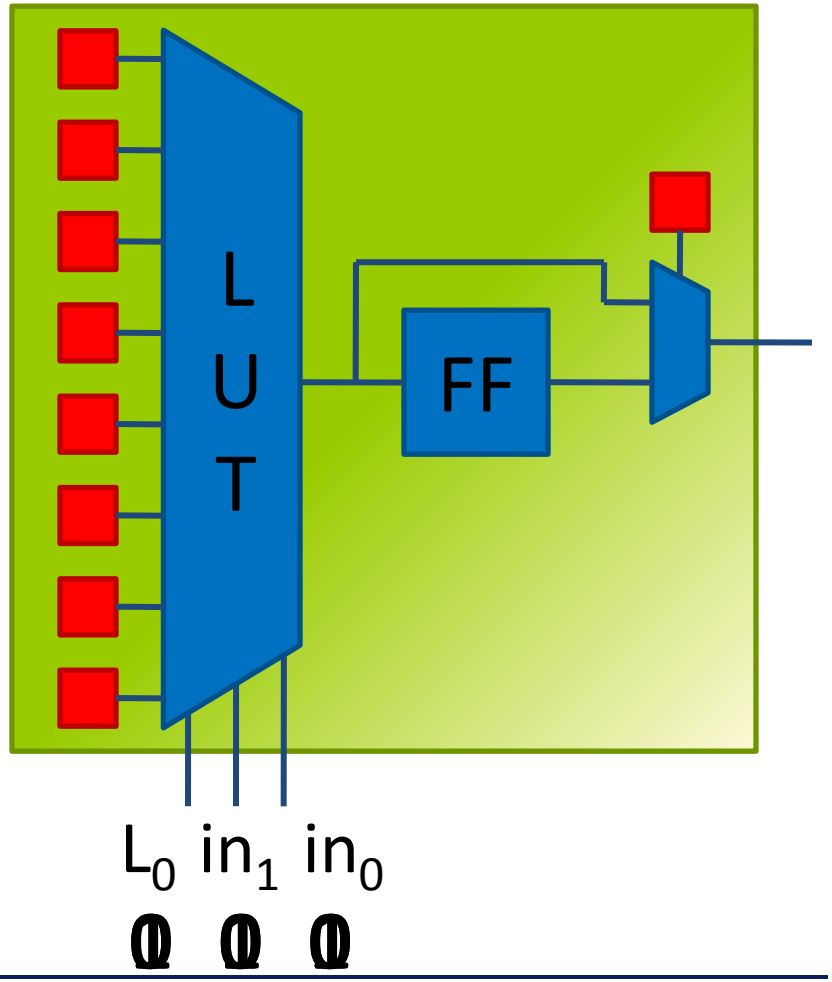
$\overline{sel_1}$

sel_1

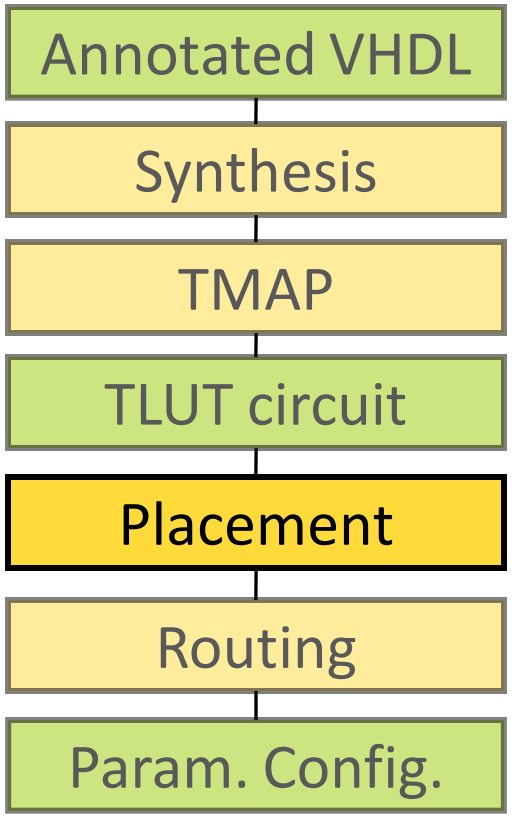
$sel_1 + \overline{sel_1}.\overline{sel_0}$

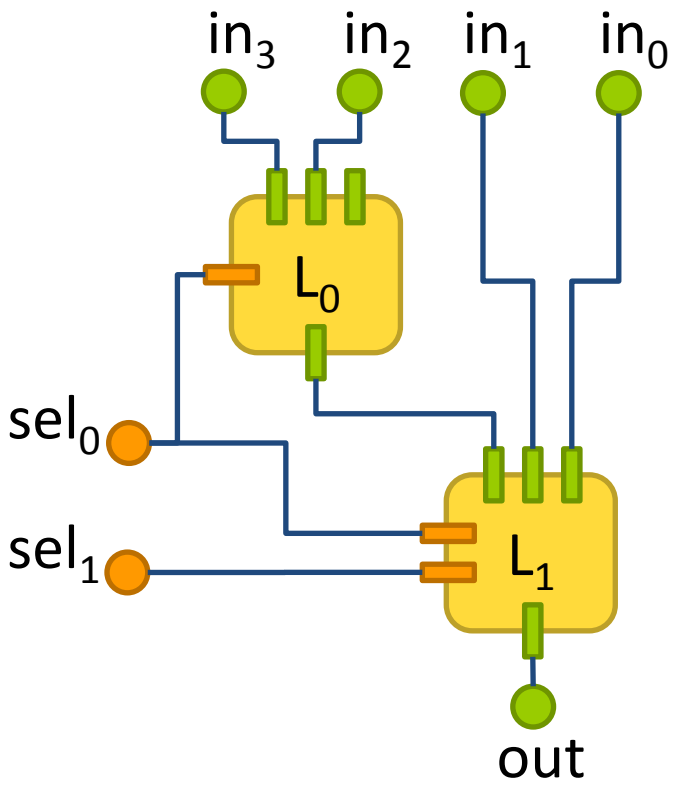
$sel_1 + \overline{sel_1}.sel_0$

1

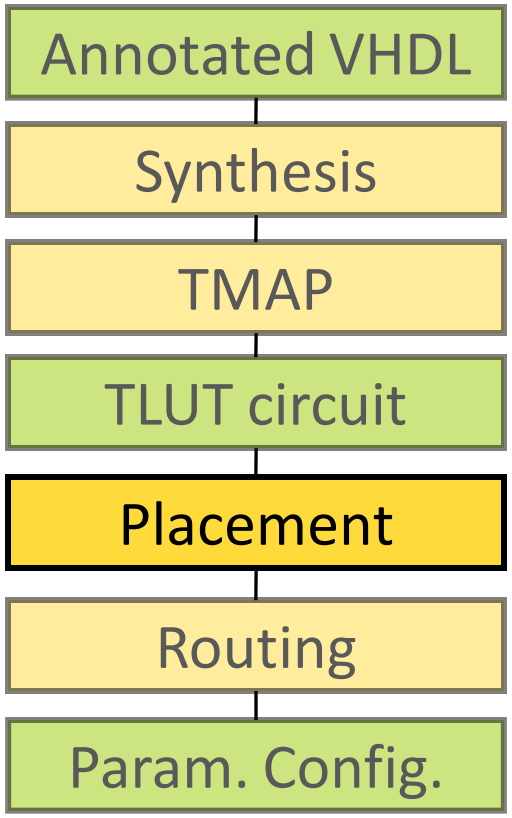


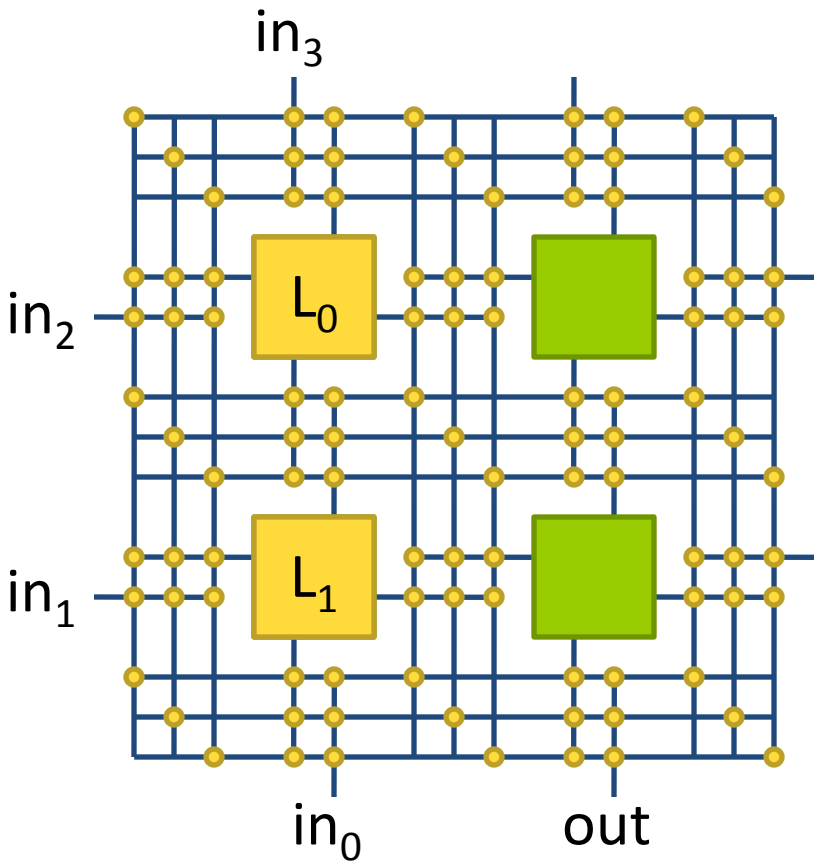
Extracting LUT circuit



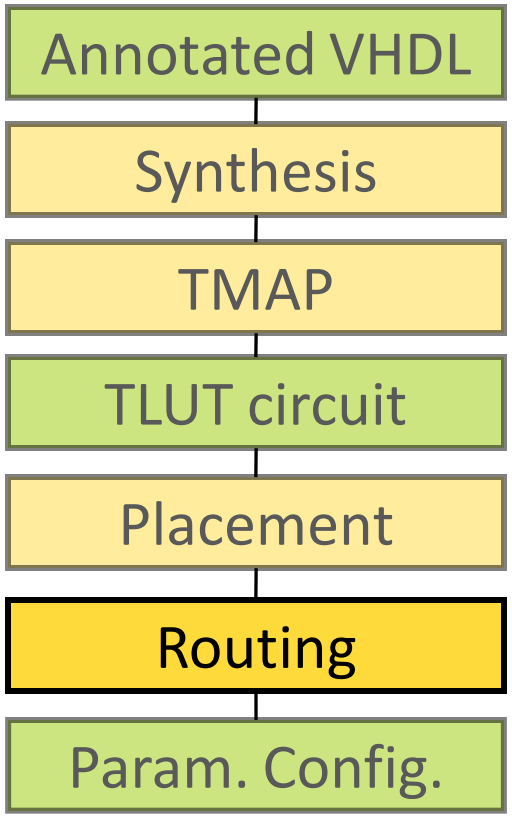


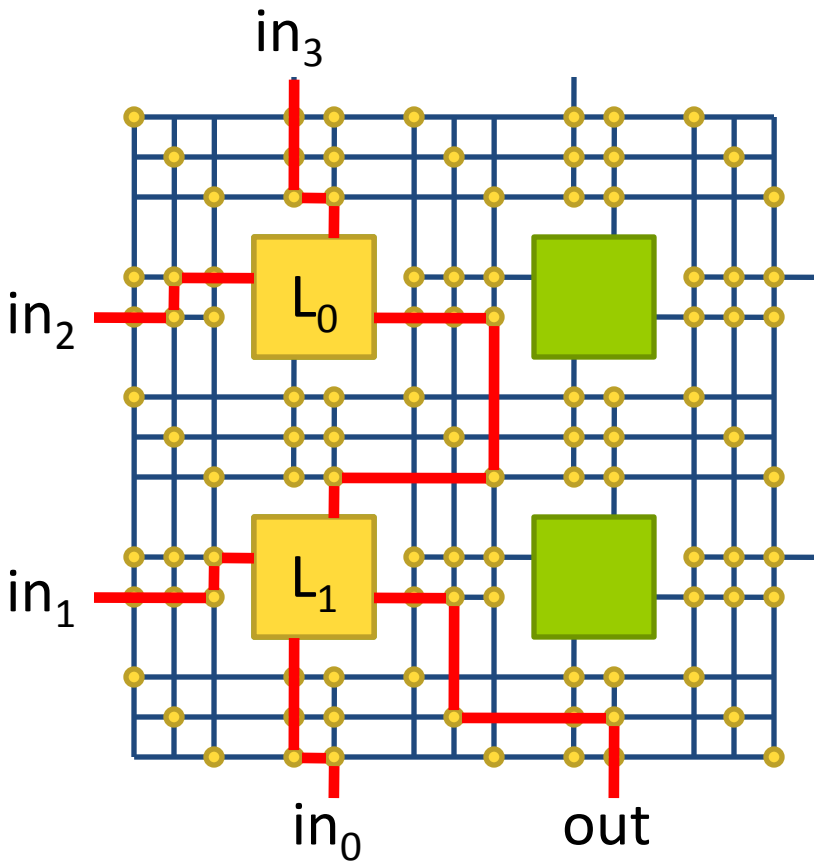
Automatic generation





Automatic generation





Parameterizable configuration

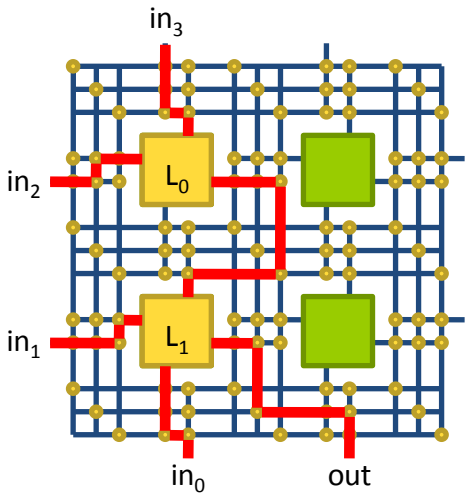
- Annotated VHDL
- Synthesis
- TMAP
- TLUT circuit
- Placement
- Routing
- Param. Config.

tuning functions

$$L_0 = sel_0.in_3 + \overline{sel_0}.in_2$$

$$L_1 = sel_1.L_0 + \overline{sel_1}.(sel_0.in_1 + \overline{sel_0}.in_0)$$

placement and routing info

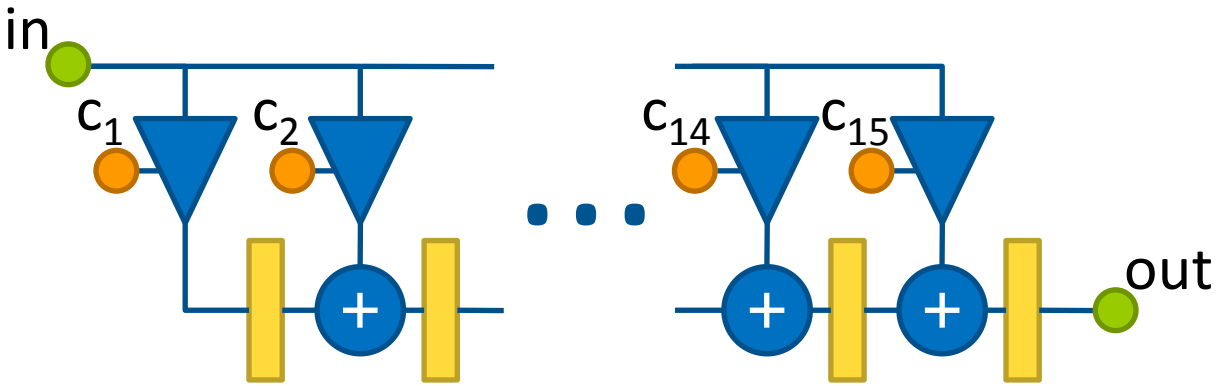


parameterizable configuration

TMAP summary

- Start from annotated VHDL
- Regular synthesis
- ***New mapping*** for parameterizable functions
- ***Reduce TLUTs*** to regular LUTs
- Perform regular placement and routing
- Parameterizable configuration:
  - ***Tuning functions***
  - Placement info
  - Routing info

Adaptive filtering experiment



- FIR filter
  - 16 taps
  - 8-bit input
  - 8-bit coefficients (parameters)
- Average over 100 coefficient sets
- Compare:
  - Off-line generated, generic FIR filter
  - On-line generated, specific FIR filter (QIS + VPR)
  - Parameterizable configuration

Experimental results

	Off-line	On-line Avg – WC	Param. config.
Area (LBs)	2999	1005 – 1147	1301
Clock freq. (MHz)	8.4	11.9 – 9.67	11.5
Gen. time (ms)	0	35634 →	0.166

Variable!

- Compared to off-line generated, generic hardware

  57% smaller - 37% faster

- Compared to on-line generated, specific hardware

  29% bigger - 3% slower (WC: 13% bigger - 19% faster)

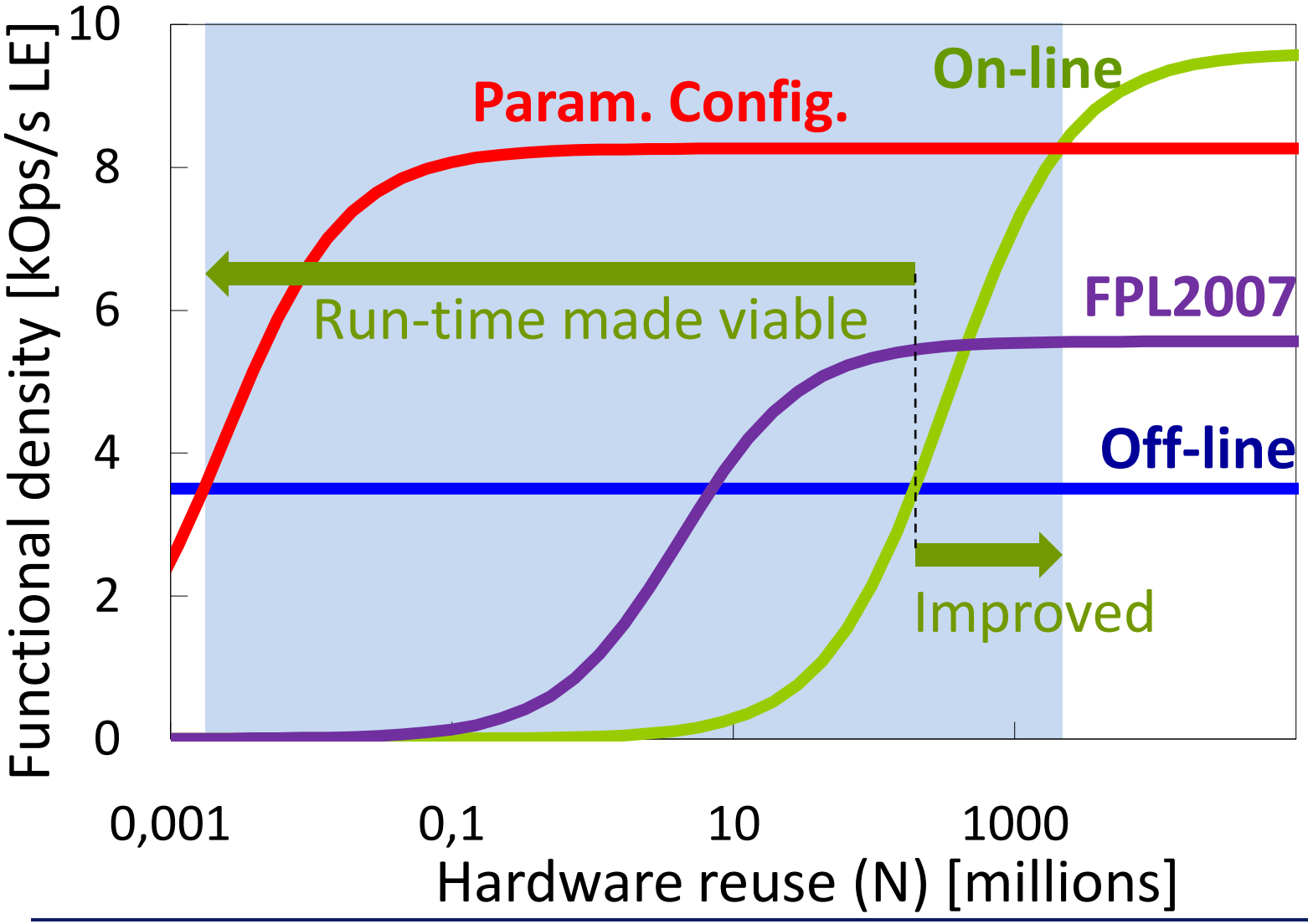
- Generation time: ***5 orders of magnitude faster***

Comparing results

- How can we compare?
  - Hardware properties: Critical path (t_{cp}) and Area (A)
  - Generation (t_{gen}) and Configuration (t_{conf}) time
  - Number of operations (N)
- Functional density: number of computations per second and per unit of area [Wirthlin et al, 1998]

$$D = \frac{N}{A(Nt_{cp} + t_{gen} + t_{conf})} = \frac{1}{A\left(t_{cp} + \frac{t_{gen}+t_{conf}}{N}\right)}$$

Functional density



Conclusions

- Parameterizable FPGA configurations
- ***Fully automatic*** method for generating parameterizable configurations, ***without extra design effort***
- Parameterizable configurations can:
  - extensively ***reduce*** hardware ***generation time***

    yet
  - produce configurations with ***good quality***

Other parameterizable problems

- Matrix multiplications
  - Multiply one vector (parameter) with all rows
- Gene sequence pair matching
  - One sequence (parameter) against full database
- Security encryption
  - Key = parameter
- …

Future work

- Improving TMAP to improve solution quality
- Introduce parameterized interconnections
- Help the designer in choosing the right parameters (performance prediction)
- Change structure of FPGAs to shorten reconfiguration time
  - Becomes the bottleneck now!
  - Part of FPGA logic could be used for evaluating tuning functions

You can help

- Other problems that would benefit from this?
- Spread the word!
- Contact us for collaborations…
  - Dirk.Stroobandt@UGent.be
  - http://hes.elis.UGent.be/

Run-time FPGA reconfiguration: From challenge to reality