Fair Comparison of Hardware Implementations of Cryptography without Revealing the Source Codes

by Kris Gaj, Venkata Amirineni,
Ekawat Homsirikamol, Marcin Rogawski,
Michal Varchola, and Rajesh Velegalati

Our Team

Venkata
“Vinny”
MS CpE
student

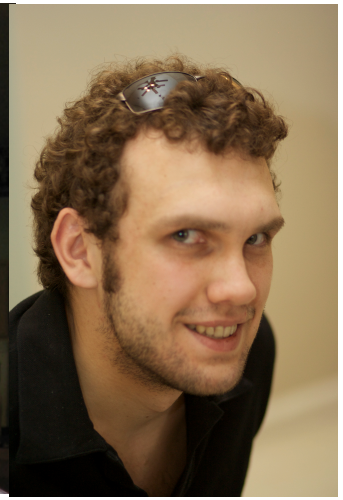
Ekawat
“Ice”
MS CpE
student

Rajesh

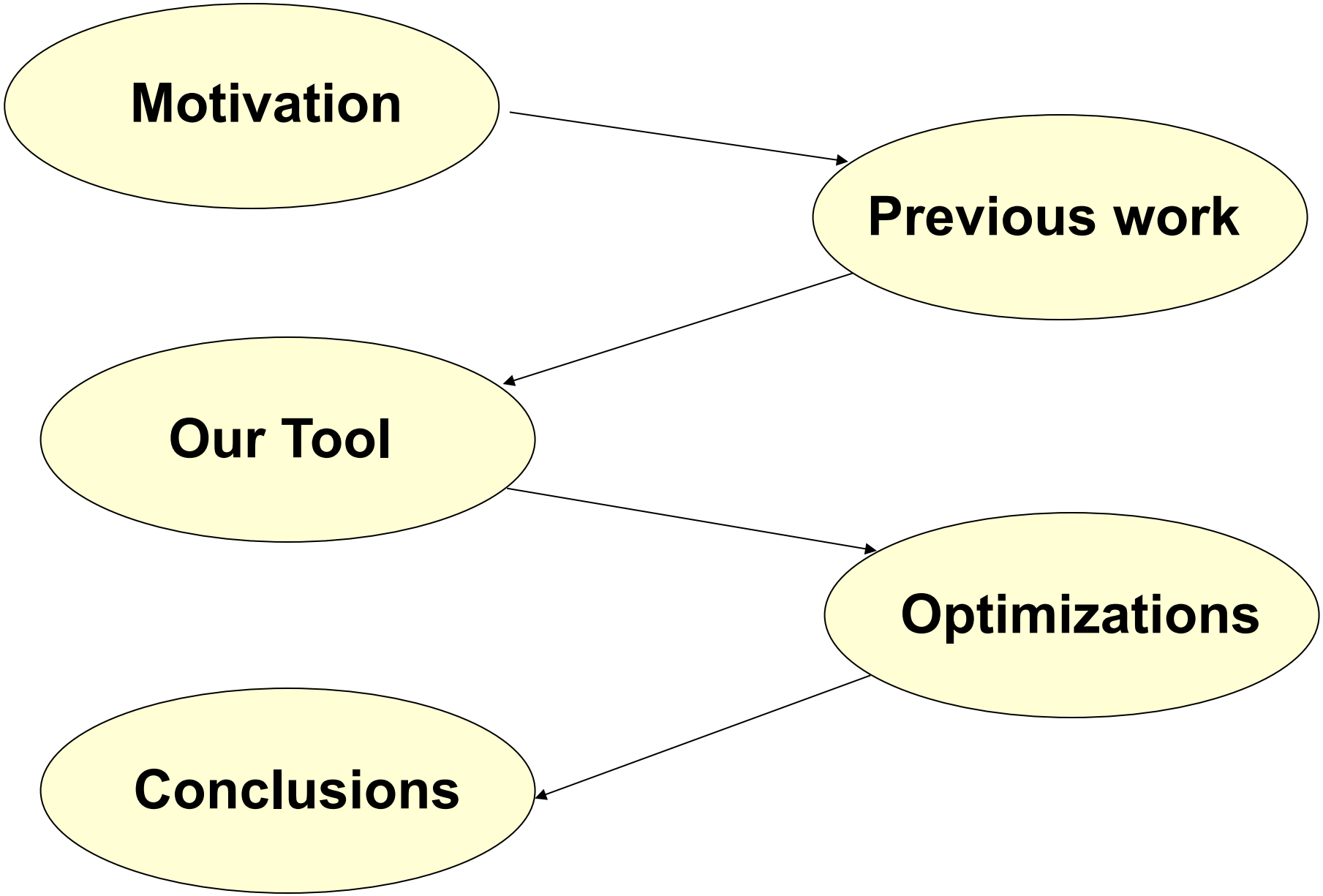
MS CpE
student

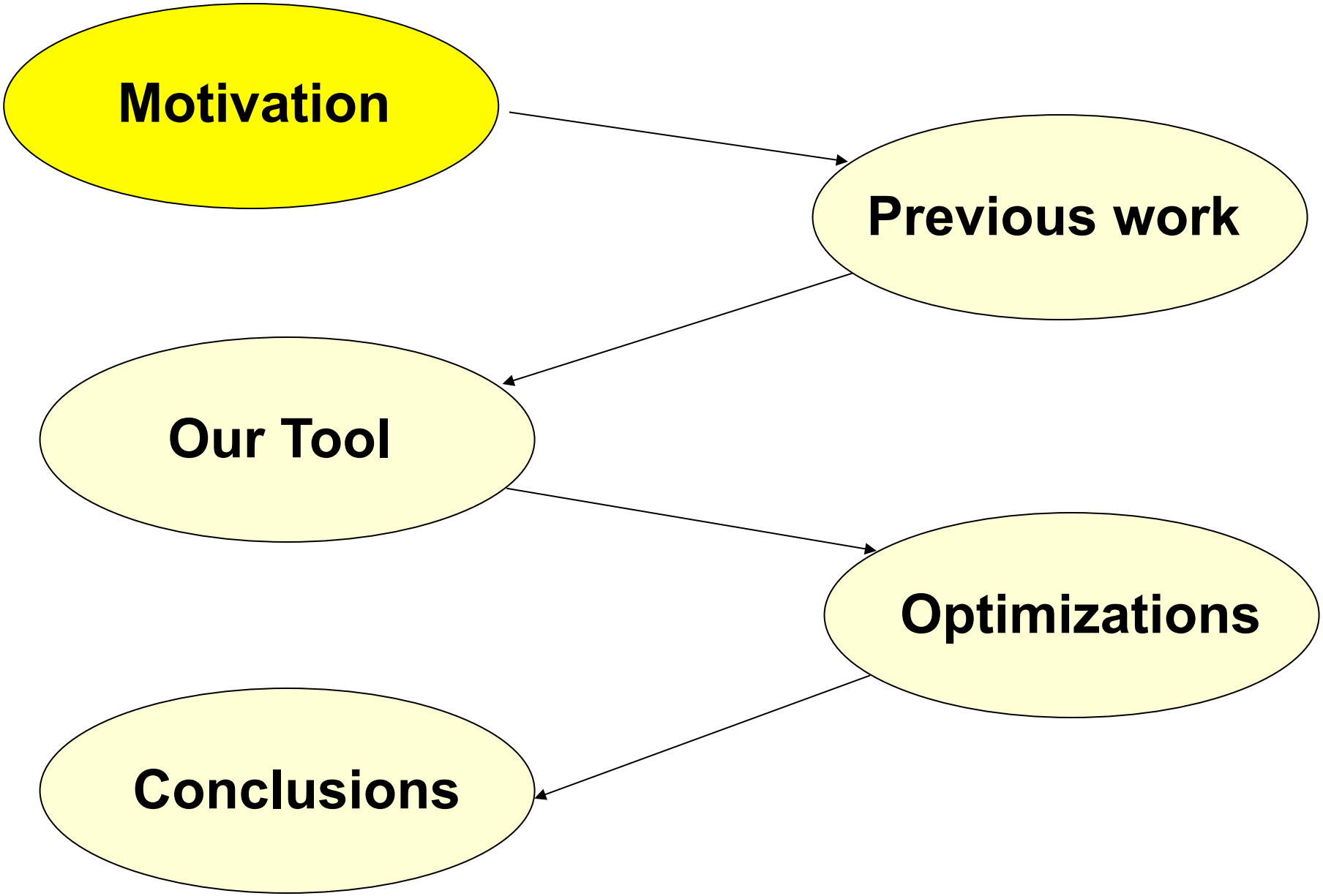
Marcin

PhD ECE
student



Michal
visitor
from Technical
University of Kosice
PhD student
of Milos Drutarovsky


Motivation
Previous work
Our Tool
Optimizations
Conclusions


Motivation
Previous work
Our Tool
Optimizations
Conclusions

Problem to be solved

Comparison of hardware implementations of cryptographic algorithms that is

- **fair** : based on objective criteria
- **comprehensive** : based on multiple hardware platforms and software tools
- **reliable** : reproducible
- **does not require revealing the code** : practical, acceptable for majority of designers

Goals

1. comparing multiple cryptographic **algorithms** competing in the contests for national and international standards, such as SHA-3 contest
2. comparing multiple hardware **architectures or implementations** of the same cryptographic algorithm, such as a paper for CryptArchi or CHES
3. comparing various hardware **platforms** from the point of view of their suitability for the implementation of a given algorithm, such as a choice of an FPGA device or FPGA board for implementing a particular cryptographic system
4. comparing various **tools and languages** in terms of quality of results they generate (e.g. Synplicity Synplify Pro vs. Xilinx XST, Verilog vs. VHDL vs. AHDL), ISE v. 10.2 vs. ISE v. 9.1, etc.)

Common research "frauds" (1)

- taking credit for improvements in technology
  e.g. comparing Bob's AES in Virtex 5 vs. Alice's AES in Virtex 2 Pro
- choosing a convenient performance measure
  e.g. Throughput/CLB slices, with majority of logic implemented using Block RAMs or DSP blocks
- comparing designs with different functionality
  e.g., encryption+decryption vs. encryption only;
  encryption+key scheduling vs. encryption only,
  three-keys-in-one vs. 128-bit key only;
  full functionality vs. core functionality with precomputations and/or postcomputations in software;
- comparing the speed of different operations
  e.g., comparing the combined speed of hashing 8 messages in parallel vs. the speed of hashing a single long message.

Common research “frauds” (2)

- designs optimized using different optimization criteria
  e.g., comparing Bob's design optimized for minimum latency
  vs. Alice's design optimized for a minimum product of latency times area (especially using latency as the only criterion)
- comparing clock frequency after synthesis vs. clock frequency after placing and routing [rare, but still happens]
- using different input/output interfaces
  e.g. 64-bit data bus vs. 32-bit data bus

Objective difficulties & possible solutions (1)

- no standard **interfaces**

Solutions:

  - develop standard interfaces for most common cryptographic operations (encryption/decryption, hash function, MAC generation/verification, etc.) or at least for most common algorithms (AES, 3DES, SHA-1, SHA-512, etc.)
  - report and reuse interfaces using the on-line database
  - compare only with designs using the same or very similar interface

- result dependent on option **settings**, amount of **time** spent on working with the tools, and the **methodology** used

Solutions:

  - propose the optimization methodology suitable for majority of designs
  - allow sufficient time in the batch mode to go over multiple tool options
  - allow designers to use the best optimization methodology they can come up with but require or at least encourage publishing such methodology

Objective difficulties & possible solutions (2)

- **implementation-specific constraints** such as path-specific timing constraints; area constraints generated using a floorplanner; physical synthesis, etc.

  Solutions:

  - compare designs in different categories without implementation-specific constraints, and with implementation-specific constraints
  - report results in both categories in order to evaluate the influence of implementation-specific constraints
  - require or at least encourage reporting constraint files

- codes may be **optimized for one particular platform** (FPGA family)

  Solutions:

  - results reported separately for each pair vendor-family
  - designer can specify a target FPGA device his design was optimized for (if any)
  - other designers can optimize their designs for the same target

Objective difficulties & possible solutions (3)

- **different tools**,
  e.g., Synplicity Synplify Pro vs. Xilinx XST

Solutions:

  - generate results for various tools
  - clearly report tools used
  - compare either only with results generated by a given tool; or with the best results obtained across different combinations of tools

- **different versions** of tools
  e.g., Xilinx ISE v. 9.1 vs. Xilinx ISE 10.2

Solutions:

  - investigate the difference in results
  - rerun the implementations periodically when the new versions of tools appear

Objective difficulties & possible solutions (4)

- influence of **surrounding logic** and **device utilization**,
  e.g., stand-alone design vs. a functional module of the bigger entity

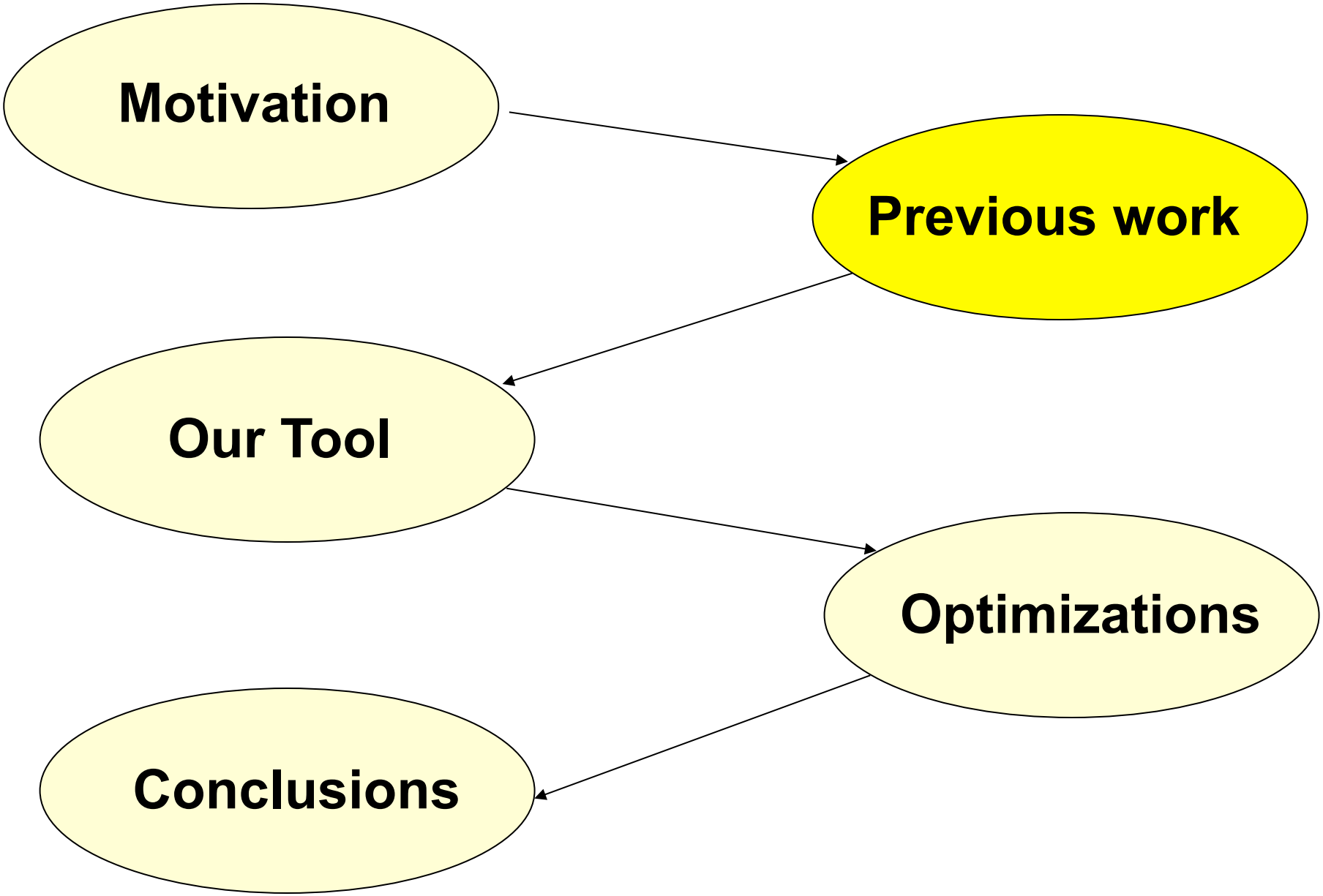
Solutions:

  - report only stand-alone designs
  - automated choice of an FPGA device aimed at a certain maximum utilization of FPGA resources (CLB slices, Block RAMs, DSP Blocks, etc.)
  - maximum utilization chosen safely based on experiments with multiple cryptographic cores

- different **optimization criteria**, such as speed, area, speed/area

Solutions:

  - always report the optimization criteria
  - compare only designs optimized using the same criterion


Motivation
Previous work
Our Tool
Optimizations
Conclusions

Previous work

eBACS: ECRYPT **B**enchmarking of **C**ryptographic **S**ystems
http://bench.cr.yp.to

Project to compare **software implementations** of cryptographic algorithms

Developed by: Daniel J. Bernstein and Tanja Lange (2006-present)

Activity of: VAMPIRE: Virtual Application and Implementation REsearch Lab

Integrates:

- eBATS: ECRYPT Benchmarking of Asymmetric Systems
- eBASC: ECRYPT Benchmarking of Stream Ciphers
- eBASH: ECRYPT Benchmarking of All Submitted Hashes

Extends earlier software evaluation projects developed by different groups within **NESSIE** and **eSTREAM**.

SUPERCOP

System for **U**nified **P**erformance **E**valuation **R**elated to **C**ryptographic **O**perations and **P**rimitives

- toolkit developed by the VAMPIRE lab for measuring the performance of cryptographic software
- measures the performance of
  - hash functions
  - secret-key stream ciphers
  - public-key encryption systems
  - public-key signature systems
  - public-key secret-sharing systems
- output is an extensive set of measurements in a form suitable for easy computer processing

SUPERCOP

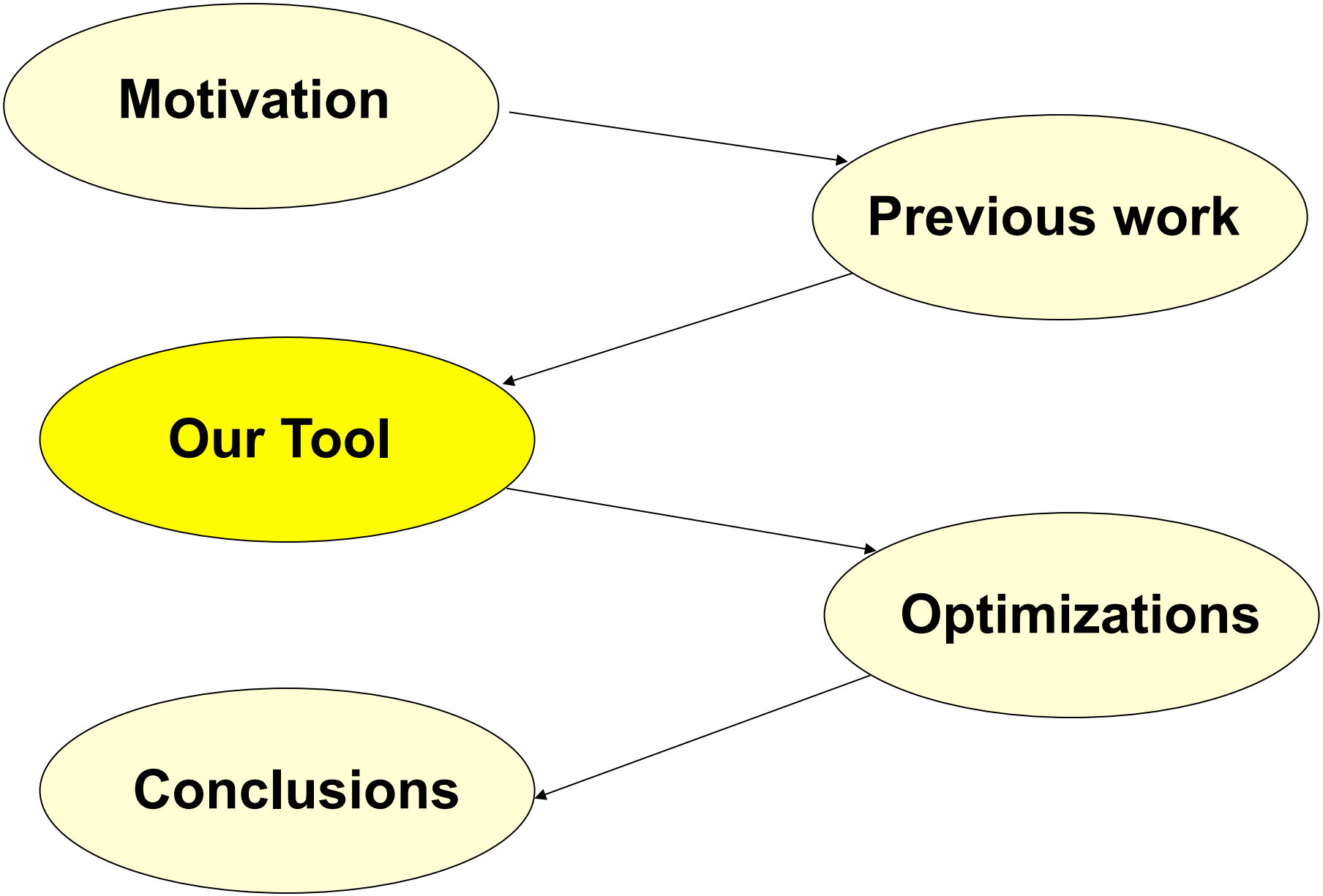
- measurements on multiple machines (currently over 70) and machine-ABI (application binary interface) combinations (currently over 100)
- each implementation is recompiled multiple times (currently over 1200 times) with various compiler options to identify best working options for implementation, machine
- time measured in clock cycles/byte for multiple input/output sizes
- median, lower quartile (25th percentile), and upper quartile (75th percentile) reported
- standardized function arguments (may be implemented using wrappers)

Similarities in comparing software and FPGA designs

- relatively few major vendors
  - Intel and AMD for general-purpose microprocessors
  - Xilinx and Altera for FPGAs
- good quality tools available for free
  - GNU compilers for software
  - full or slightly reduced versions of tools for FPGAs
- multiple options of tools
- software programs can be written targeting a specific processor;
  HDL codes can be written targeting a specific FPGA family
- low level optimizations possible but typically not portable:
  in software - assembly language; in FPGAs - low level macros

Differences in comparing software and FPGA designs

- in software speed is a major parameter;
  in hardware **speed and area** need to be taken into account and can be often traded one for the other
- in software clock frequency is fixed for a given processor;
  tools try to optimize the sequence of instructions;
  in FPGAs **clock frequency** determined by the implemented circuit;
  tools try to optimize the most critical paths, and thus minimize the clock period
- in software execution time is **measured** directly with some non-negligable measurement error; in FPGAs minimum clock period is **reported** by software tools; minimum execution time is calculated;
- **open source** software cryptographic libraries widely available; very few open source cryptographic hardware designs


Motivation
Previous work
Our Tool
Optimizations
Conclusions

Our Tool

ATHENa – **A**utomated **T**ool for **H**ardware **E**valuatio**N**

Set of scripts written in Perl aimed at an
AUTOMATED generation of
OPTIMIZED results for
MULTIPLE hardware platforms

Currently under development at
George Mason University.
The first proof-of-concept version
to be released before
CHES 2009, and announced
during the rump session at CHES.

Our Tool

ATHENCA – **A**utomated **T**ool for **H**ardware **E**valuatio**N** of **C**ryptographic **A**lgorithms

Focus on comparison of cryptographic **algorithms**, such as hash functions competing in the SHA-3 contest

Still young and immature…

ATHENa Major Features

- running all steps of synthesis, implementation, and timing analysis in the batch mode
- support for devices and tools of multiple FPGA vendors:
  Xilinx, Altera, Actel
- generation of results for multiple families of FPGAs of a given vendor
- automated choice of a device within a given family of FPGAa assuming that the resource utilization does not exceed a certain limit, e.g., 80% of CLB slices, or 70% of BRAM
- choice of multiple optimization criteria (speed, area, ratio speed/area)
- heuristic optimization algorithms aimed at maximizing the performance measures (e.g., speed) based on checking multiple options, and multiple target clock frequencies

ATHENa Additional Features

- automated verification of the design through simulation, run in the batch mode based on the provided testbench (optional):
  - Functional
  - Post-synthesis
  - Timing
- support for Windows and Linux
- Graphical User Interface

Requirements:

- interpreter of Perl
- FPGA tools: free, educational, or commercial versions

ATHENa Input/Output

Input:

- synthesizable source files
- configuration files (text files)
- testbench (optional)
- constraint files (optional)

Output:

- result summary (human readable)
- database entries (suitable for computer postprocessing)

Design Configuration File

PROJECT NAME = AES128

TOP_LEVEL_ENTITY = aes_top

OPTIONS = [default | user]

OPTIMIZATIONS = [single_run | default | user]

VENDOR = [Xilinx | Altera | Actel]

FPGA_FAMILY = [Spartan3 | Virtex4 | Virtex5 | etc.]

FPGA_DEVICES = [best_match | list of specific_devices]

#for best_match and Xilinx FPGAs only

MAX_CLB_UTILIZATION = 0.8

MAX_BRAM_UTILIZATION = 1.0

MAX_MUL_UTILIZATION = 1.0

MAX_PIN_UTILIZATION = 0.9

Device library file

VENDOR = Xilinx

FAMILY = Spartan3

DEVICES =

#Device,	Total CLBs,	Block RAMs,	Dedicated Multipliers,	Max. User I/O
XC3S50,	728,	4,	4,	124
XC3S200,	320,	12,	12,	173
XC3S400,	896,	16,	16,	264
XC3S1000,	1920,	24,	24,	391
XC3S1500,	3328,	32,	32,	487
XC3S2000,	5120,	40,	40,	565
XC3S4000,	6912,	96,	96,	633
XC3S5000,	8320,	104,	104,	633

Tool Configuration File

SYNPLIFY_DIR = C:/synpro/bin

XILINX_DIR = [C:/Xilinx91i/bin/nt | C:\Xilinx\10.1\ISE\bin\nt]

ISE_VERSION = [9.1 | 10.1]

SYNTHESIS_TOOL = [synplify | native]

Result Summary (1)

---Tool names and versions---

ISE web pack 9.1

---Optimizations---

ATHENa default

---Family and Device---

Spartan3 xc3s50pq208-5

Virtex4 xc4vsx25ff668-12

---Synthesis options---

-opt_mode speed

-opt_level 1

Result Summary (2)

Timing results:

Devices	Synthesis	Implementation
xc3s50pq208-5	56.899 MHz	40.455 MHz
xc4vsx25ff668-12	98.260 MHz	98.600 MHz

Resource utilization:

Devices	CLB Slices	BRAMs	Multipliers	DSP blocks	IOpins
xc3s50pq208-5	459	0	0	0	77
xc4vsx25ff668-12	459	0	0	0	77

Basic Dataflow of ATHENa

User

FPGA Synthesis and Implementation

6

5

Database query

Ranking of designs

2

3

HDL + scripts + configuration files

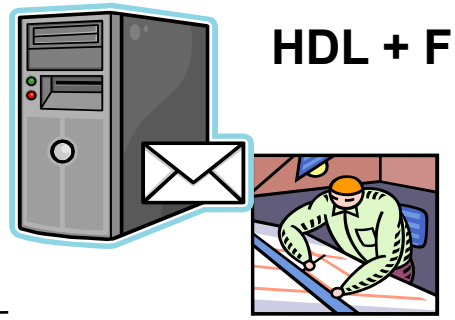
Result Summary + Database Entries

ATHENa Server

1

Download scripts and configuration files

HDL + FPGA Tools



4

Database Entries

Designer

Polices regarding the submission of results

Require:

- tool names, versions, detailed options

Encourage:

- I/O Interface
- Testbench
- Constraint files

Why?

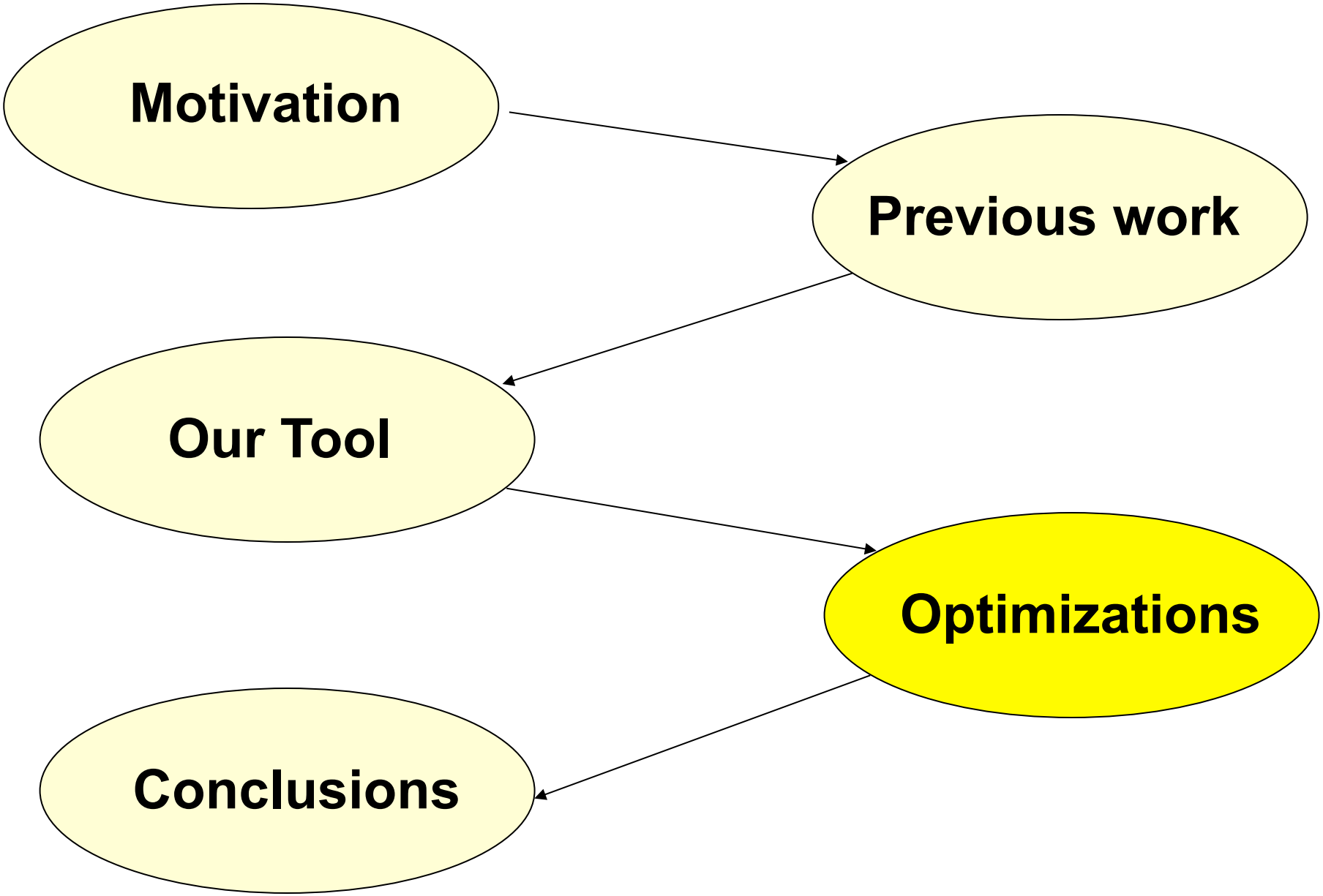
- do not reveal much information about the internal structure of codes
- Save time of subsequent designers
- Significantly simplify fair comparison

Optional:

- source code a plus, but not really crucial for comparison

ATHENa Reporting

- majority of database entries generated automatically by the scripts (based on configuration files and result summaries)
- some fields generated through postprocessing (e.g., latency and throughput); it is the designer's responsibility to provide the correct formula
- data retrieved from the database based on the selected fields
  e.g., algorithm: AES
  key size: 128
  FPGA vendor: Altera
  all other fields: any
- data arranged based on a value of the specific field
  e.g., by Throughput *or* by Number of CLB slices
- previously reported results can be replaced with the new ones or completely withdrawn by the users


Motivation
Previous work
Our Tool
Optimizations
Conclusions

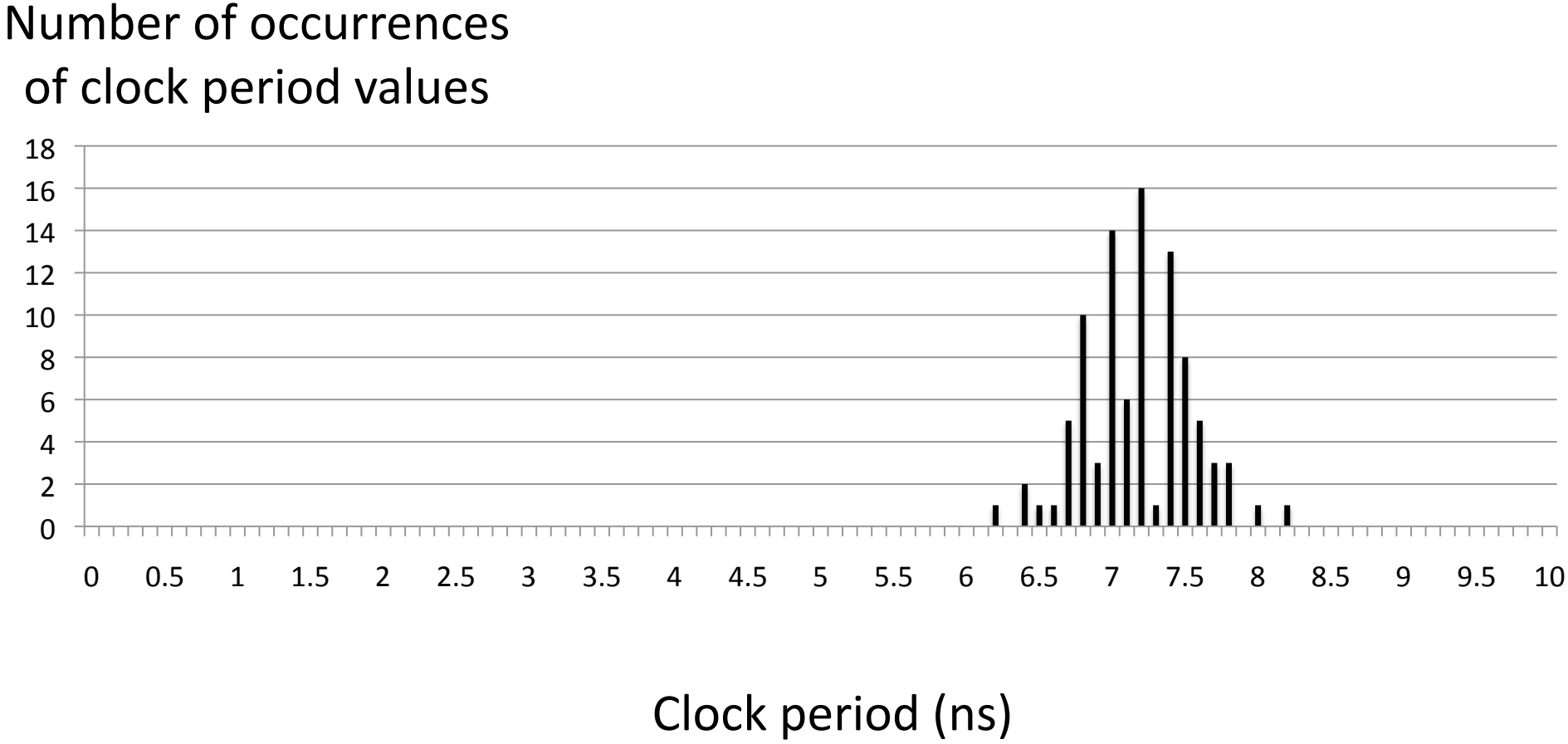
Optimization Algorithm

- Multiple optimization algorithms considered and tested experimentally using available cryptographic cores.
- Separate algorithm required for different optimization criteria:
  Speed, Area, Speed/Area.
  Speed always proportional to Clock Frequency .
  Area may have different measures (CLB slices, LUTs, BRAM, MULs, etc.)
- One or more algorithms to be selected in each category as default algorithms, with corresponding scripts distributed as a part of ATHENa.
- Designers are encouraged to develop and implement their own optimization algorithms. Designers encouraged to post description or implementation of these algorithms together with their results to make them available to other members of the community.

First Optimization Algorithm by Marcin Rogawski

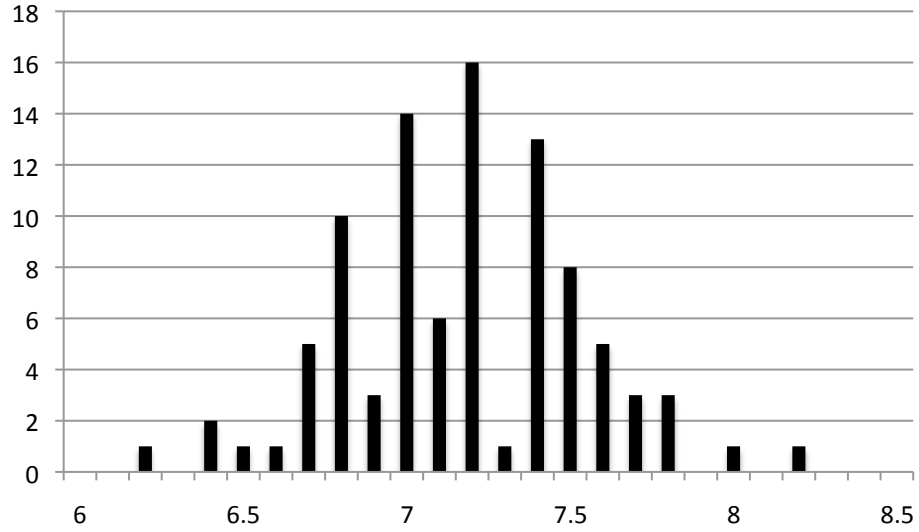
- first single run with default options of tools to determine the initial value of the target clock frequency
- requested clock frequency increased gradually, first by 5%, and then (after failure) by 1%
- for each requested clock frequency: up to 100 runs of Multi-Pass Place-and-Route with different values of a cost table (determining the starting point for placement), until the requested frequency reached
- synthesis repeated for each new requested clock frequency
- optimum choice of the overall, placer, and router efforts

Multi-Pass Place-and-Route Analysis

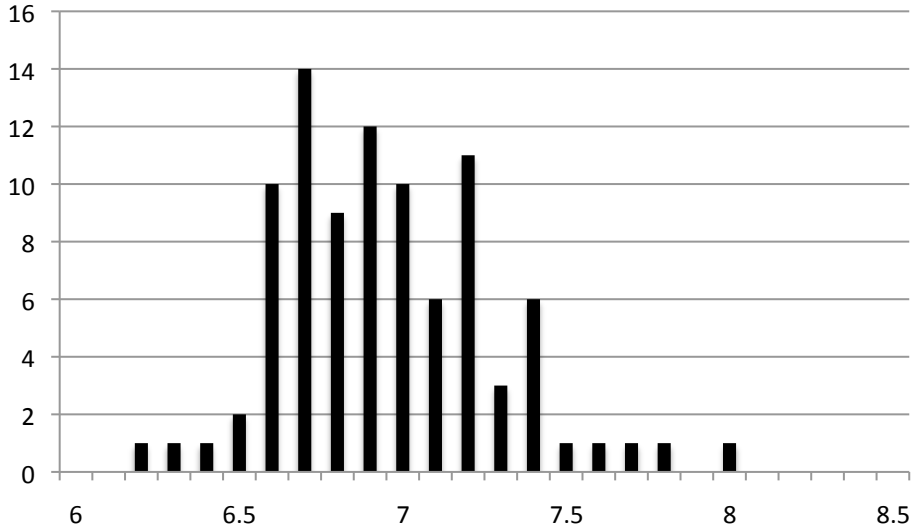


Dependence of results on requested clock frequency

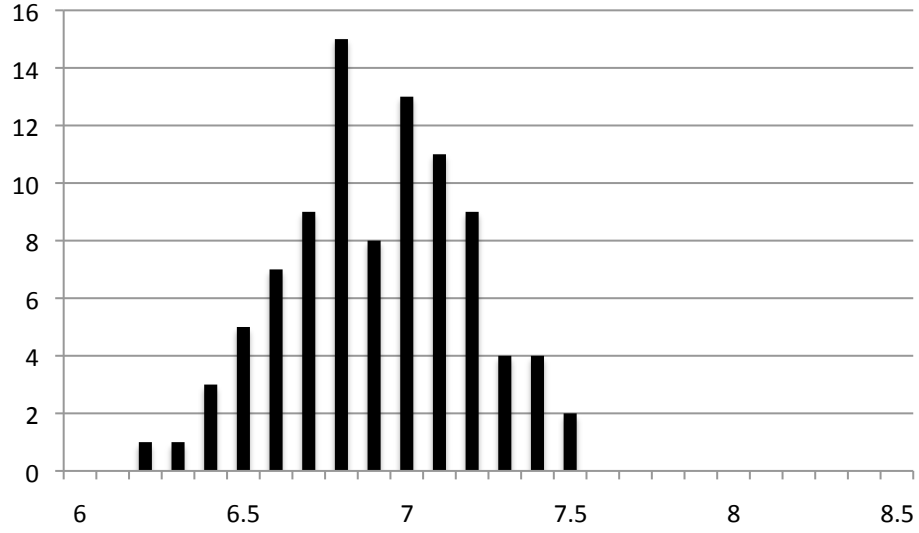
400 MHz



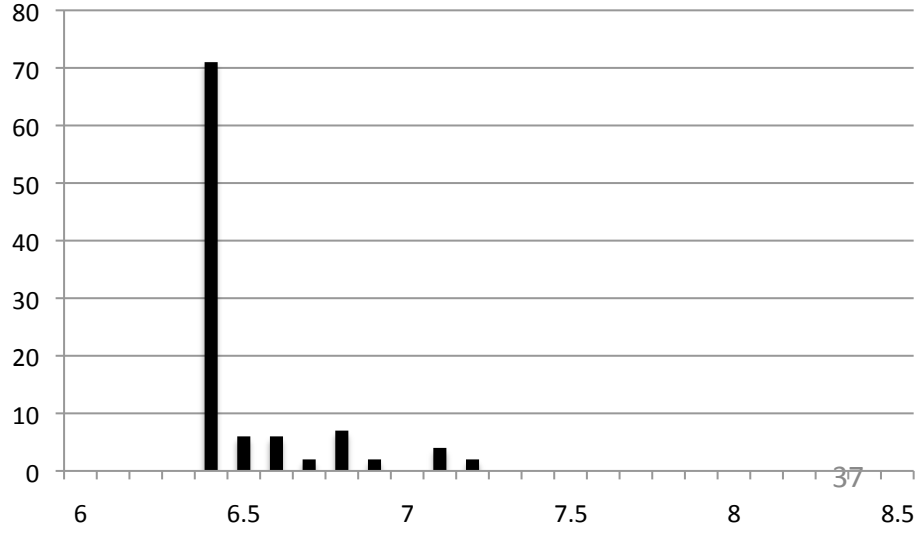
350 MHz



300 MHz



ATHENa optimization algorithm



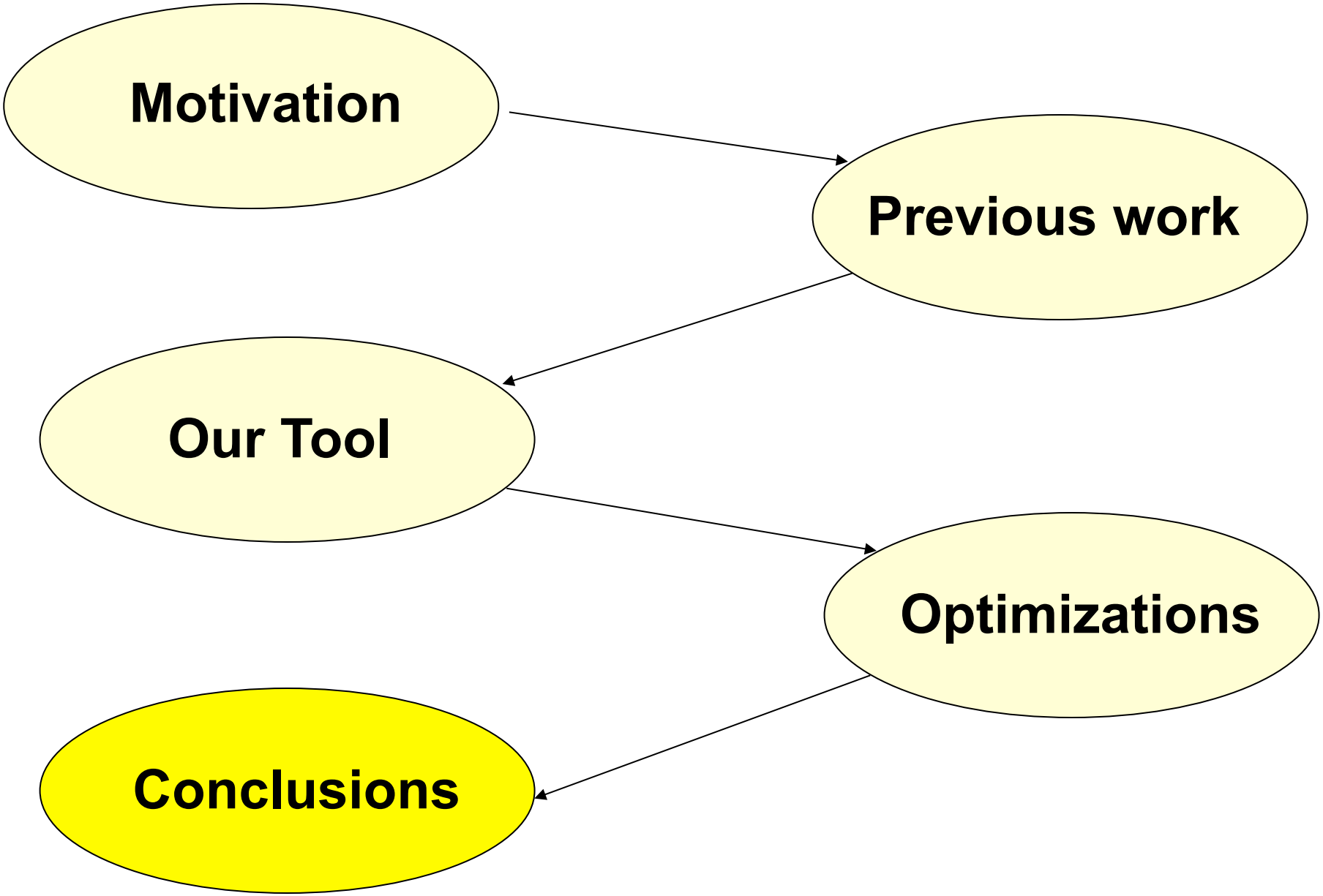
Optimization Type

- Type 1: **default vendor** options of tools (single run)
- Type 2: **ATHENa default** optimization algorithm with ATHENa default configuration options
- Type 3: **designer's own** optimization algorithm or configuration options **without floorplaning** or physical synthesis
- Type 4: **designer's own** optimization algorithm involving **floorplaning** or physical synthesis
- Type 5: designer's own optimization algorithm involving **manual routing**

Most of the users expected to submit results of Type 1 and 2 only.

Most-advanced users would submit results of Type >= 3.

Users submitting the results of Type N requested to submit also results for all Type numbers < N.


Motivation
Previous work
Our Tool
Optimizations
Conclusions

Fairness is in the eyes of the beholder

The best tools will not replace:

- proper motivation and internal integrity of researchers
- proper supervision by faculty advisors and senior managers
- thorough evaluation by independent and objective reviewers
- proper system of awards and sanctions

Possible extensions

- standard-cell ASICs
- actual experimental measurements in hardware (power and energy consumption, latency, throughput)
- taking into account resistance to side-channel attacks
- other fields (e.g. DSP)

= material for additional Master's and PhD Theses

Conclusions

- We propose a tool for a fair, comprehensive, reliable, and practical evaluation of cryptographic hardware, without the need to reveal the source code.
- Hope to discourage naive and/or dishonest comparisons, provide transparency, and overcome objective difficulties.
- First version expected at the end of summer. Subsequent versions made available as the tool matures.
- All scripts and configuration files will be made available in public domain through the project web site.
- Comments, feedback, use, and co-development very welcome.

Questions?

Comments?