FASTER toolchain: Reconfiguration Aware Mapping

Riccardo Cattaneo, Gianluca Durelli, Christian Pilato,
Marco Rabozzi, Fabrizio Spada, Alberto Scolari,
Marco Domenico Santambrogio and Donatella Sciuto

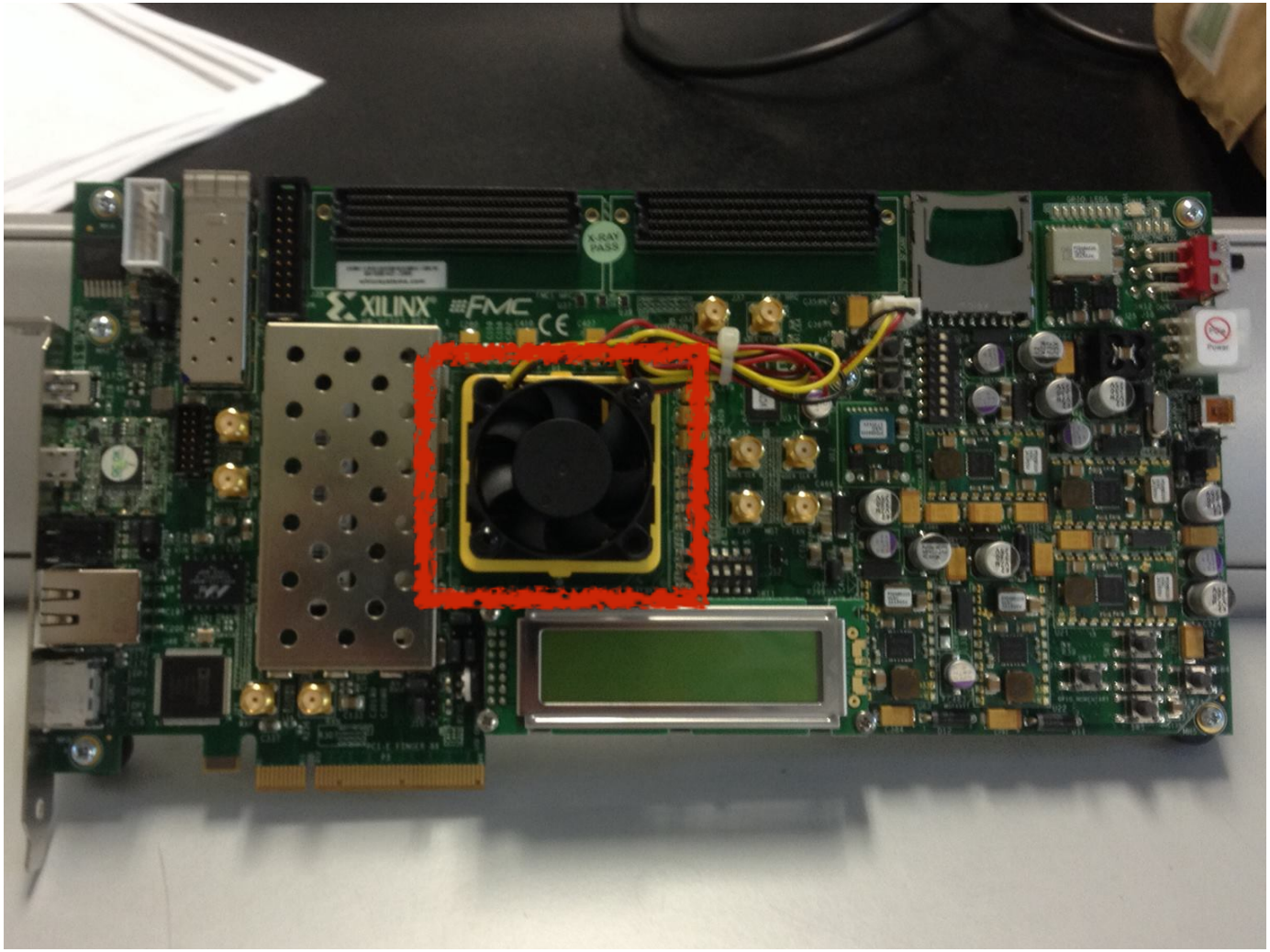
Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
Milano, IT

Politecnico di Milano
Aug 29, 2014, Milano, IT

Hi!

Riccardo Cattaneo
- 2nd year PhD student @ NECSTLab
- Reconfigurable architectures for HPC systems
- Currently working on FASTER and exaFPGA projects

What is this?



A necessary premise: the FPGA / 1

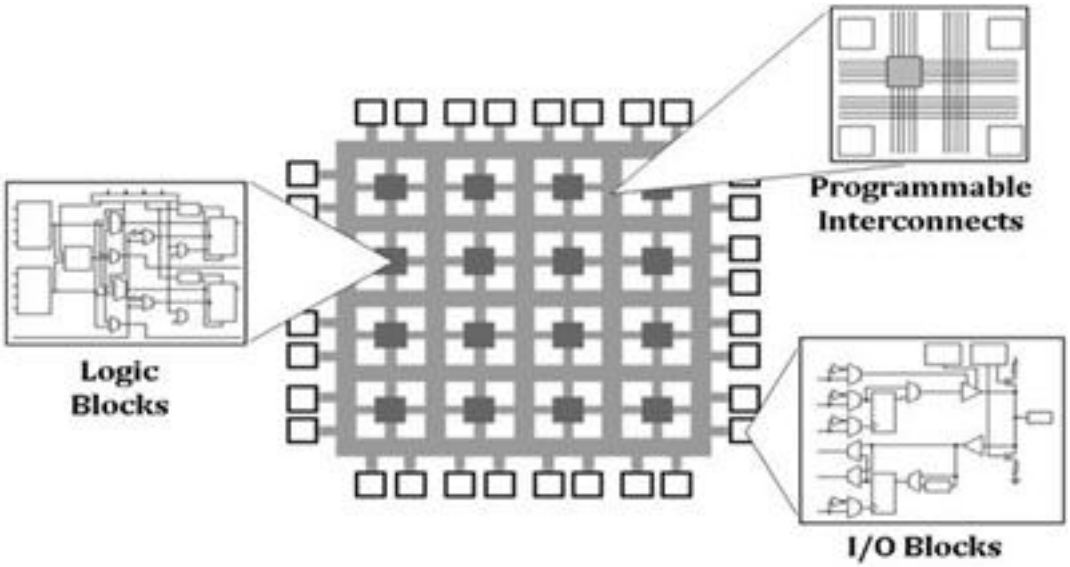
- What is an FPGA?
  - **FPGA**: **F**ield **P**rogrammable **G**ate **A**rray
  - It is an **hardware device** on which it is possible to **configure and reconfigure** an application specific **digital** (potentially, mixed signal) **circuit**
  - It is typically designed as a **non homogeneous grid of interconnected components**
    - look-up tables (LUTs), block rams (BRAMs), digital signal processors (DSPs), switch matrices, input/output blocks (IOBs) etc…
  - Roughly speaking, the interconnection among these components can be **programmed and reprogrammed** in order to **realize a specific function** (in the form of a digital circuit)
  - **Flexibility at hardware speed** (**not quite** ASIC, however!)
  - **Parallelism at hardware level** (depending on application)
    - Hardware is "**intrinsically**" running in parallel on the device
  - **Run-time reconfiguration potentially** allows for extremely efficient and flexible designs

A necessary premise: the FPGA / 2

- The device is used in many different contexts
  - **Telco** (digital circuits built around high speed transceivers for high speed digital communications)
  - **Finance** (real time estimation of the risk of a portfolio of financial instruments)
  - **Hardware and computer engineering** (emulation of hardware components, in hardware)
  - **Scientific computing** (among the others acceleration of physical systems in geology, 3D computer graphics rendering)
  - **Aerospace/Defense** (missiles, avionics, MILCOM)
  - **Medical** (MRI, PET, Intuitive Systems' Da Vinci minimally invasive surgery system)

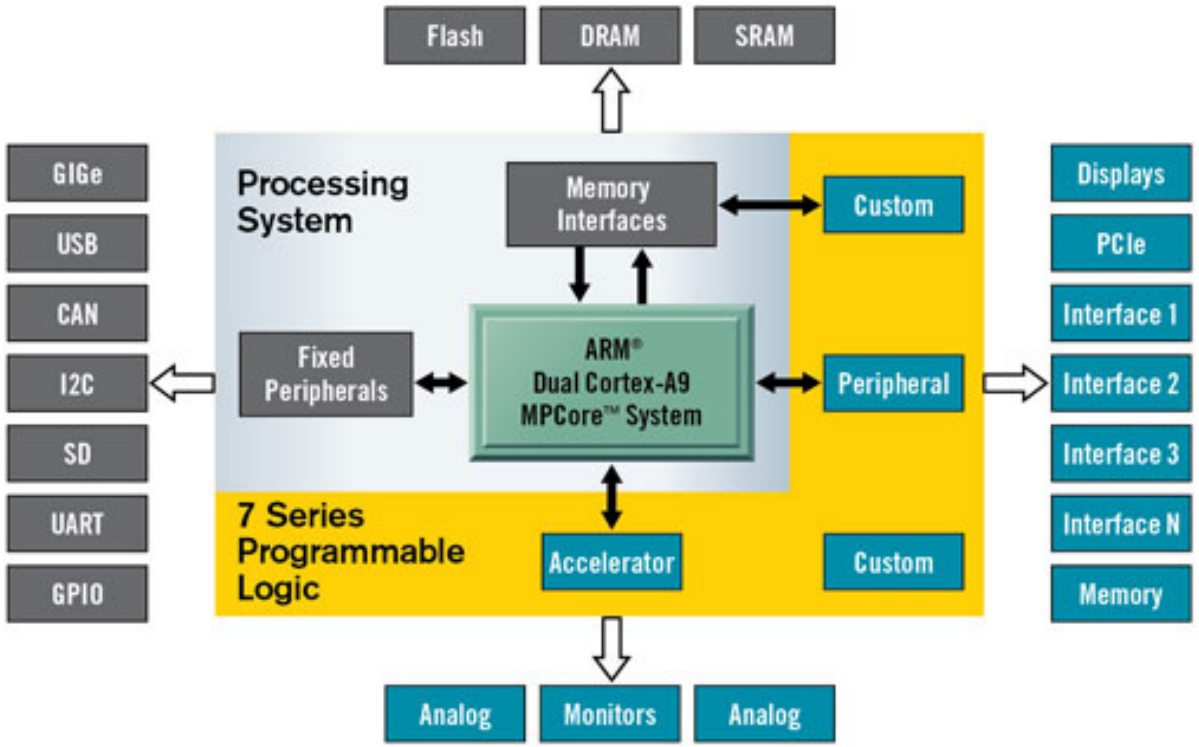
A necessary premise: the FPGA / 3

- Working with an FPGA: a rough design flow
  - The hardware engineer describes the required functionality in a Hardware Description Logic (**HDL**) language
  - The functionalities are combined in larger functionalities
  - This description is **synthesized** (~compiled) into a **digital circuit**
  - This circuit is realized by means of
    - Adequately configured **logic blocks**…
    - …connected among the others via **programmable interconnects**…
    - …and connected to the outside world via **I/O blocks**



A necessary premise: the FPGA / 4

- The foreseeable future of FPGA: highly coupled **heterogeneous** system
  - **Zynq Platform**: ARM Dual-Cortex A9 (ASIC!) cores on-die tightly coupled with a 7series (i.e.: the currently latest tech) programmable logic
  - **High speed, low latency** reconfigurable interconnect



Coarse Grain overview of Zynq7000 All-Programmable SoC

AVNet ZedBoard
(Zynq7000-based dev board)

- A little premise
- Problem statement & opportunities
- The FASTER approach
  - Motivation
  - Methodology and framework overview
    - ACO-based mapper
    - Static scheduler and runtime manager
    - Floorplacer
    - Code generation
- Experiments and results

Problem statement / 1

- In the race towards power efficiency, **Reconfigurable Hardware** has recently become an attractive platform for the development of **custom, application-specifc hardware accelerators**.

While attractive under the performance point of view, these accelerators and the relative architecture on which to execute them **are but easy to develop, verify, and run**.

For this reason, the reference reconfigurable hardware device, namely, **Field Programmable Gate Arrays (FPGA)**, is still not-so-commonly employed in production systems.

Problem statement / 2

- Additionally, FPGAs are not only reconfigurable at design-time, but also as run-time, thanks to **Partial and Dynamic Reconfiguration (PDR)**.

However, PDR is still an **untamed feature**, mostly due to the **difficulties** experienced during design time by designers and **requirement for an early planning** of its employment in a reconfigurable design.

Critical Factors

- Hardware Description Languages (**HDLs**) **are not (l)user-friendly**
  - their semantics is totally different than that implied by common programming languages such as C/C++
  - this means that the learning curve is anything but steep

An FPGA engineer must take into account **multiple constraints** at **different abstraction levels**

- **HDL**: how to write good hardware code
- **From HDL to FPGA**: lots of tools, each of which with lots of degrees of freedom impacting on the final design
- **SW**: tedious task to write “bridge” code to interface software and hardware part of the system, very error prone
- **Application-level**: how to effectively express the application so that hardware implementation is accurate and possibly straightforward
- **Verification**: complex debugging tools spanning the whole technology stack are not really mature
- **Modeling**: it is still difficult to model these systems for early validation and performance characterization of a specific design
- **Technology jails**: FPGA vendors force designers to be expert about their technology to effectively exploit them but don’t allow interoperability (but for a subset of the HDL languages)

Underlying issue: CS education is not very hardware-friendly, ECE education is not very software-friendly

- Benefits lying in this gap are not exploited yet

FASTER

FASTER aims at facilitating the design of a reconfigurable system by providing useful abstractions and an easy-to-use production toolchains to rapidly explore the impact of PDR on an FPGA-based computing system.

Partners

POLITECNICO DI MILANO

UNIVERSITEIT GENT

Imperial College London

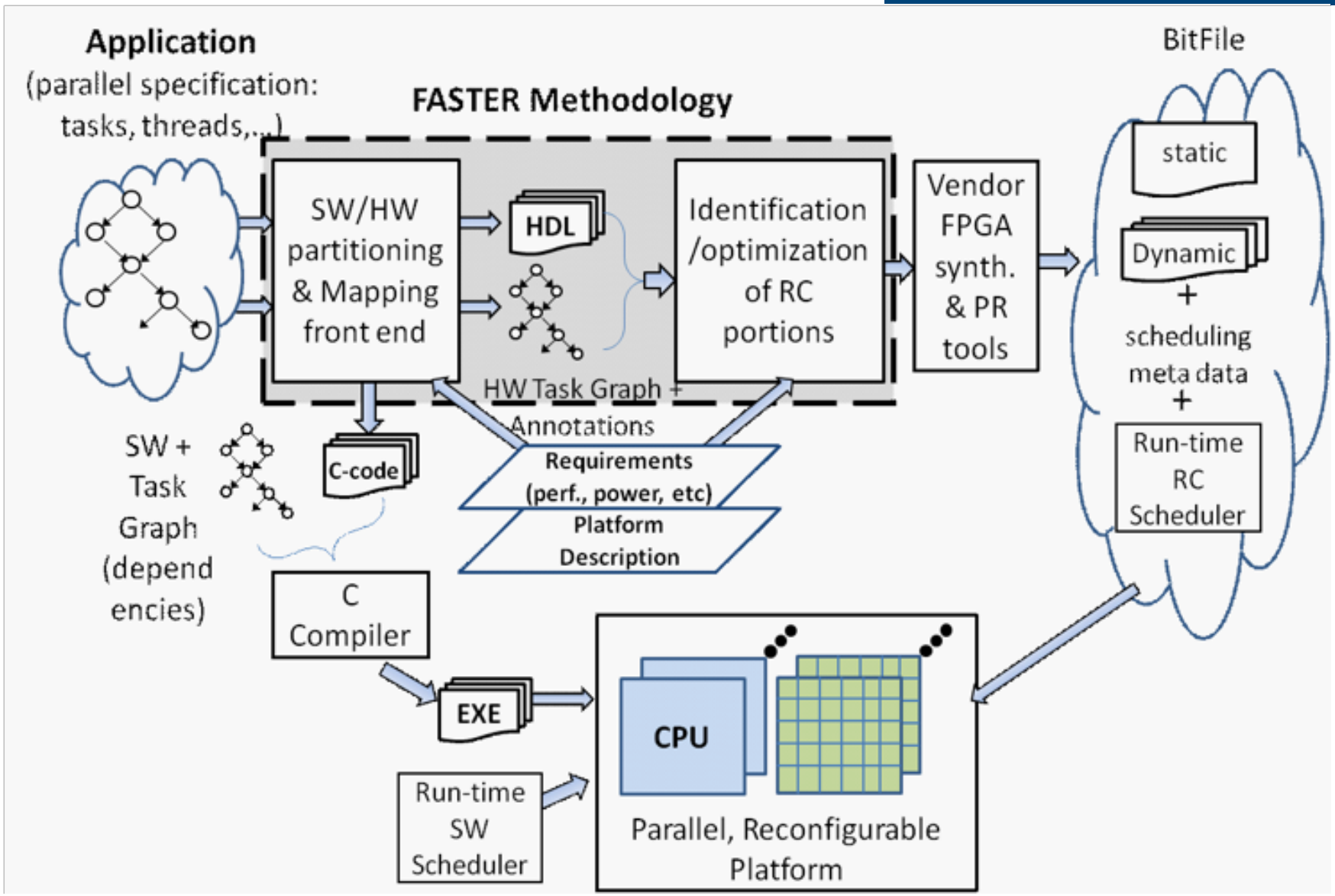
SYNELIXIS

- A little premise
- Problem statement & opportunities
- The FASTER approach
  - Motivation
  - Supported platforms and test cases
  - Methodology and framework overview
    - ACO-based mapper
    - Static scheduler and runtime manager
    - Code generation
- Experiments and results

System Analysis and Design

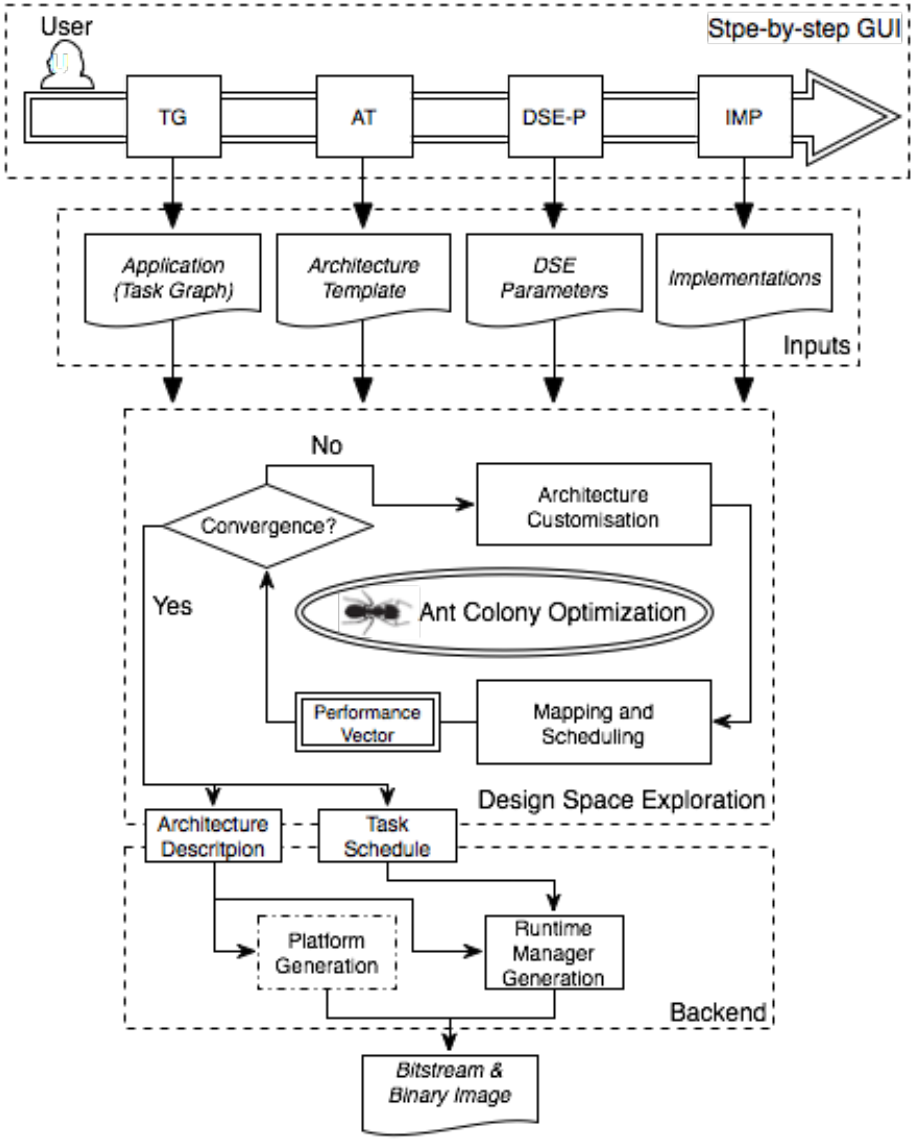
- The starting point: **the application to be ported on hardware**
- **Application Analysis**
  - Identification of components by means of software analysis
    - Kernels, static affine nested loop-based programs (SANLPs), adequate graph-based representations
  - Estimation of the performance and the constraints associated to those components on the target reconfigurable system
    - Execution time, floor planning and placement, power consumption…
- **FASTER approach**
  - Automatic **HW/SW partitioning**
  - Automatic **mapping of tasks to components**
  - Automatic **identification of partial reconfiguration opportunities**
  - Automatic **identification of reconfigurable areas**
- Refinement/Code generation step: **platform specifc backends**
  - generation of **vendor specific project** to interface with their toolchain

FASTER: overall methodology



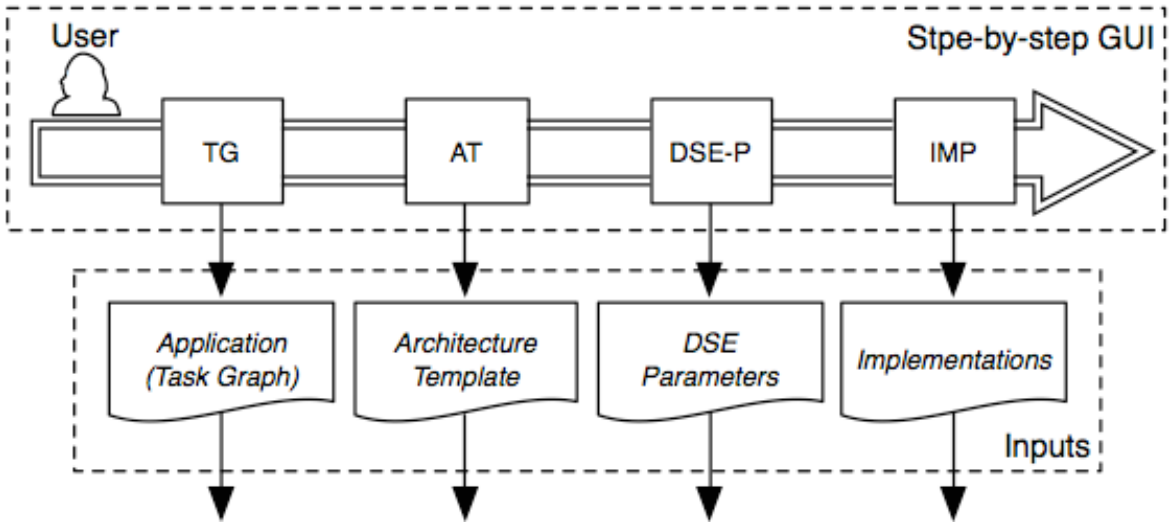
FASTER: overall framework

- **Inputs**:
  - Information about target device (.XML)
  - Application source files (.C)
- **Decision Making (Exploration)**:
  - App analysis
  - Task/Dataflow graph generation
  - Library generation
  - Mapping, Scheduling, Floor planning
  - Architectural modification
- **Refinement (Evaluation)**:
  - Specification of the platform
  - Generation of the SW code
- **Output**:
  - Project files ready for synthesis with back-end tools



FASTER: input flow

User inputs the application's taskgraph, sw and hw implementations (modes of execution with different performance profiles), the architectural template, and the design space exploration parameters



XML Exchange Format

- The entire project is represented through an **XML file**
  - Architecture: components' characteristics (e.g., reconfigurable regions), …
  - Applications: source code files and profiling information
  - Library: task implementations with the characterization (time, resources, …)
  - Partitions: task graph, mapping and scheduling, …
- It allows a **modular organization** of the framework
  - Phases can be applied in any order to progressively optimize the design
  - Designer can perform as many iterations as he/she wants to refine the solution
- Specific details of the target architecture are taken into account only in the refinement phase
  - Interactions with backend tools

Task Graph and Library Generation

- The application is written in C code and represents a set of **interacting tasks**
  - Interacting here means that each function produces and consumes data for other functions in the application
- Application source code is analyzed to extract the task graph and relevant information about the tasks based on **pragma annotations**
  - Kernel tasks (compliant with OpenMP 3.0)
  - Memory accesses and related access patterns

C.Ciobanu et al. SAMOS'13

- **Mercurium+LLVM compiler** to extract each task DFG
  - Estimation of required resources (including bit-width analysis)
  - Interaction with HLS synthesis tools for real values
    - Code rewriting for improving the synthesis (e.g., SystemC backend)
- Generated implementations are then stored in the XML file to offer opportunities to the mapping phase
  - Possibility to perform multi-objective design space exploration to generate alternative cores

C. Pilato et al. JSA'08

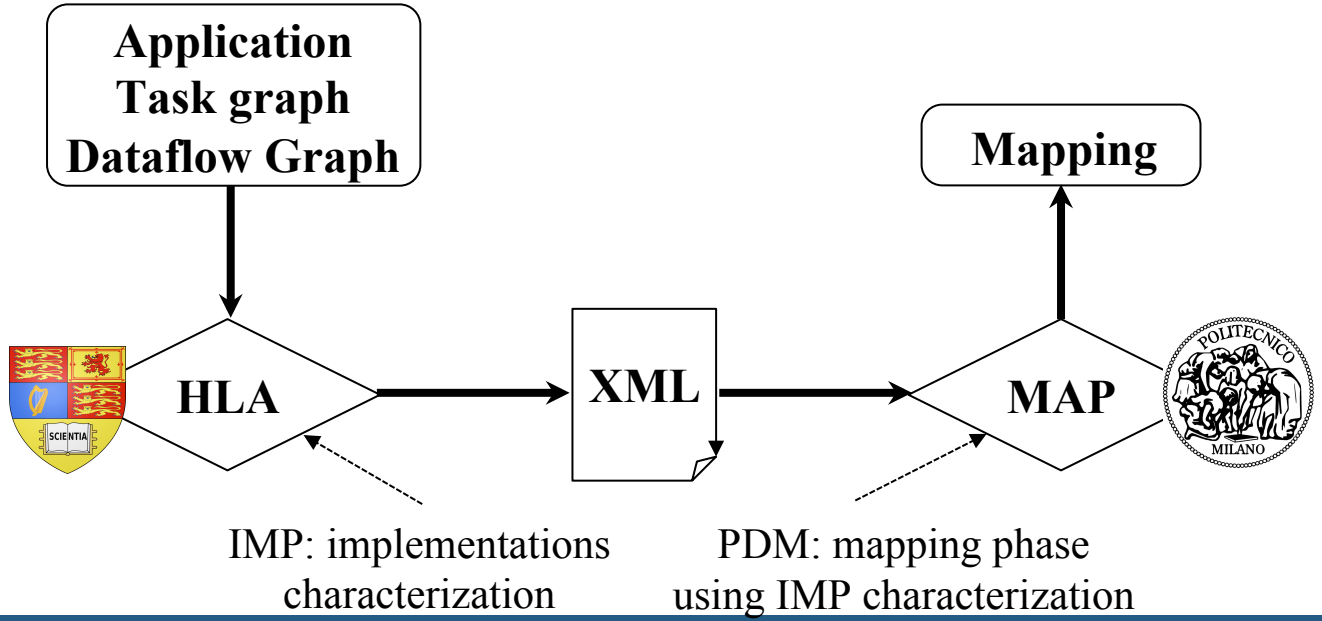
Library Generation

Library: **collection of** software and hardware **implementations**, one or more per **task** of the original task graph

We need to know how these implementations perform: need for an **estimate of resource consumption** per each task

In this work it is done via **High Level Analysis**, a fast analytic approach to estimate resources consumption of (a-fair-subset-of-) C functions

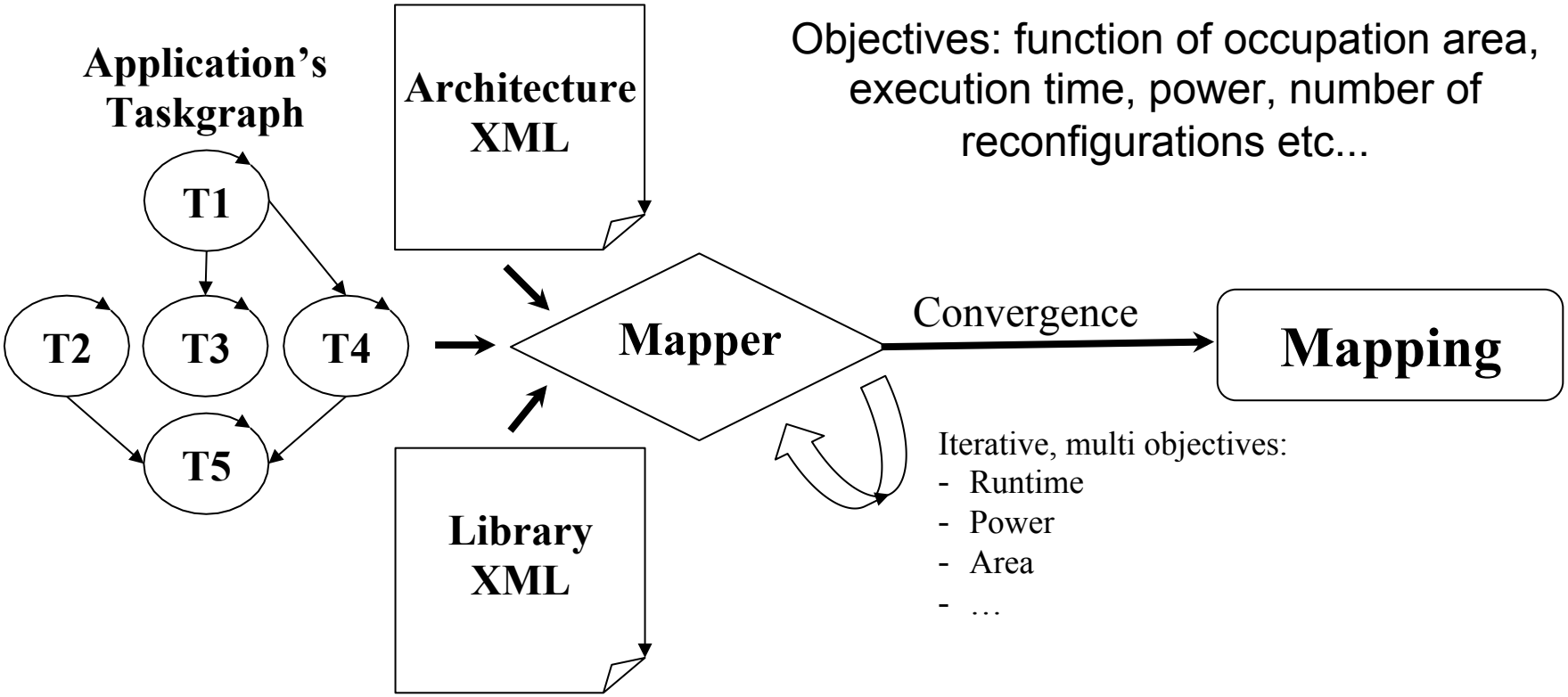
Politecnico di Milano/Imperial College of London joint effort to integrate High Level Analysis techniques into the toolchain



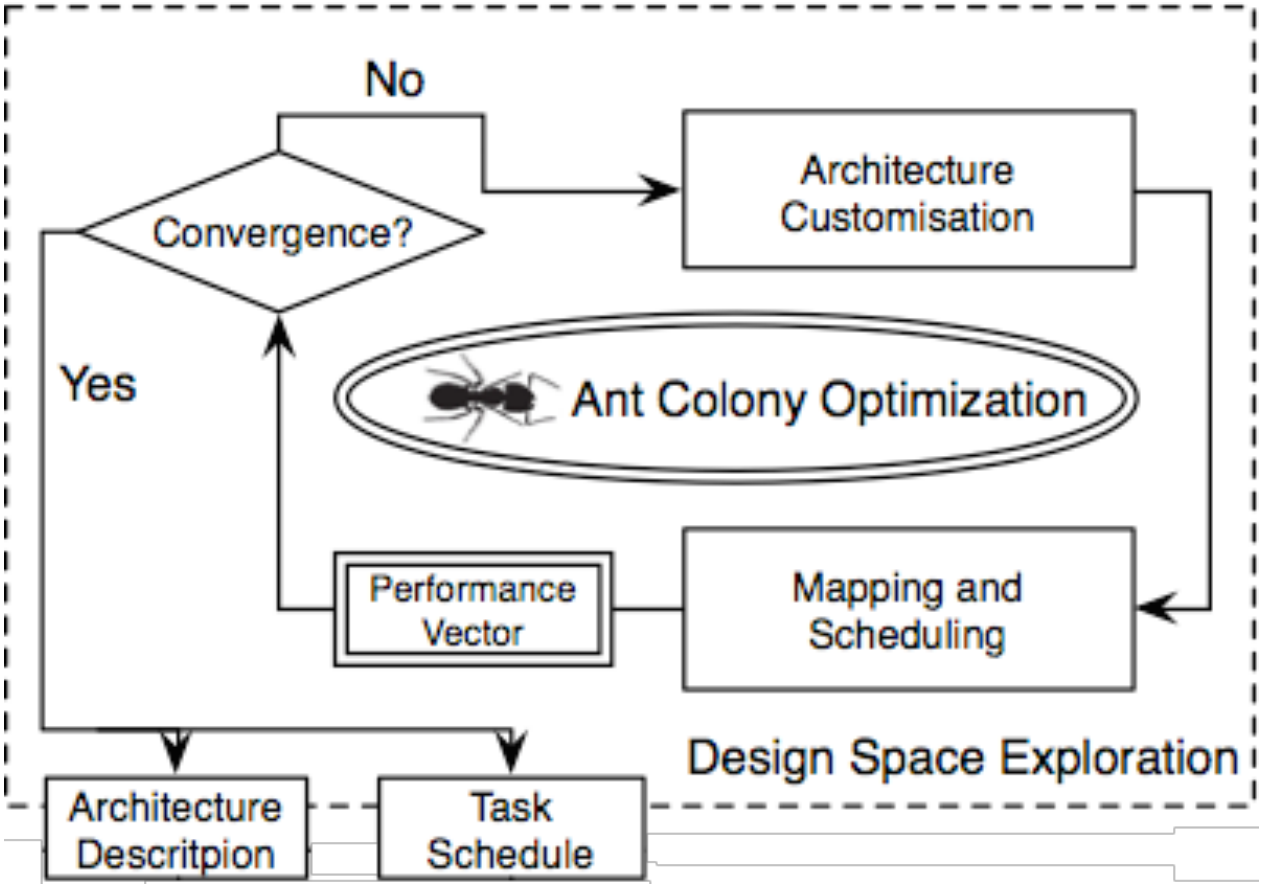
Mapping: overview

Process of **assigning each task** in the original task graph to the **"best" processor and implementation** in the system

Based on a **metaheuristic** iterative algorithm to solve a **multiobjective optimization** problem

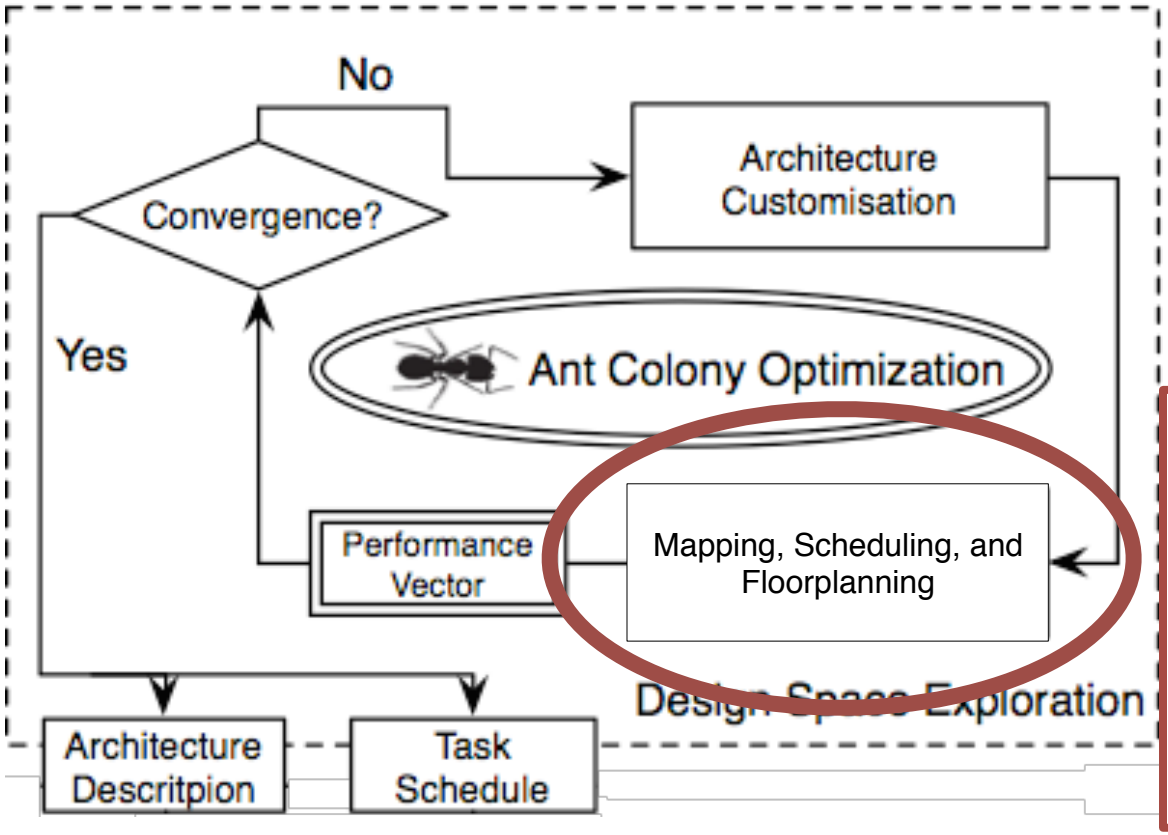


Design Space Exploration



DORIGO, Marco; BIRATTARI, Mauro. Ant colony optimization. In: *Encyclopedia of Machine Learning*. Springer US, 2010. p. 36-39.

Design Space Exploration

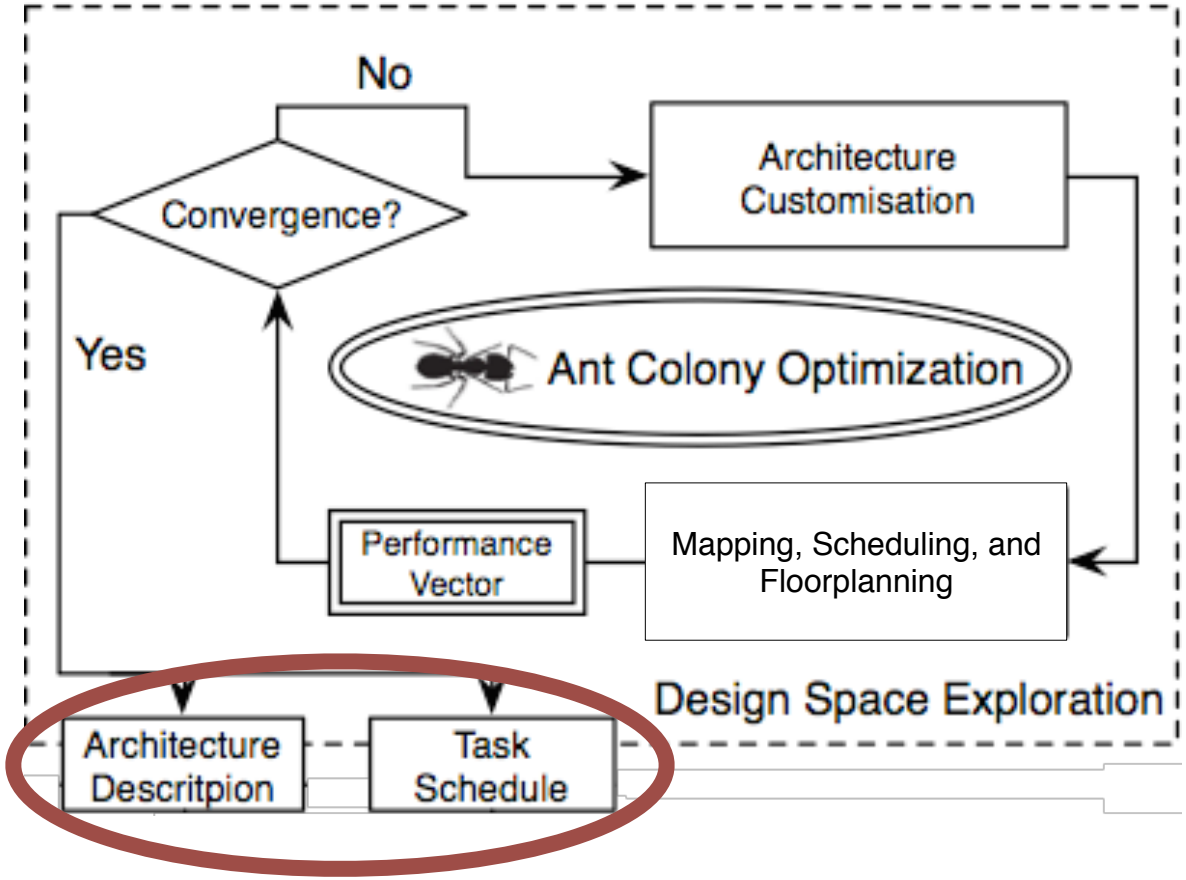


Critical step

Quality of solution depends on previous steps of the ACO algorithm

Design Space Exploration

Solution: mapping, scheduling, floorplanning, adequate architecture

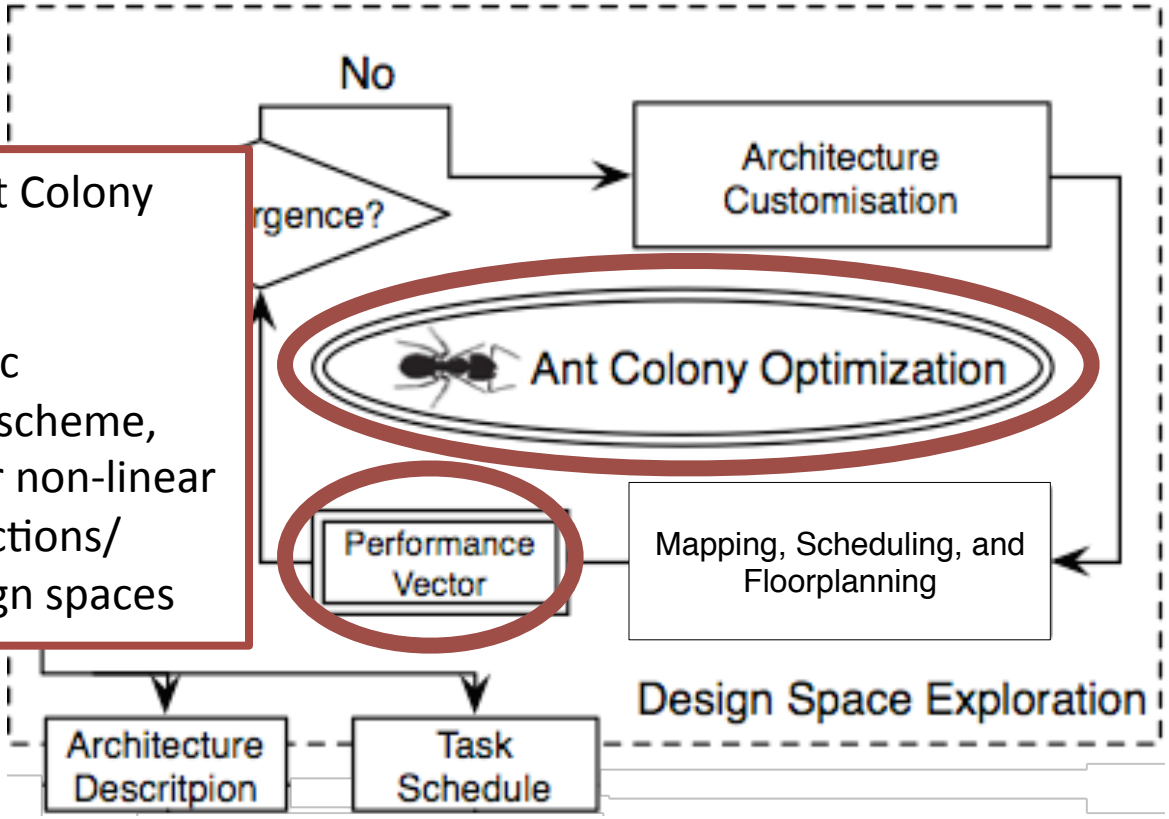


DORIGO, Marco; BIRATTARI, Mauro. Ant colony optimization. In: *Encyclopedia of Machine Learning*. Springer US, 2010. p. 36-39.

Design Space Exploration

Evolution: Ant Colony Optimization

Meta Heuristic Optimization scheme, viable also for non-linear objective functions/ complex design spaces



DORIGO, Marco; BIRATTARI, Mauro. Ant colony optimization. In: *Encyclopedia of Machine Learning*. Springer US, 2010. p. 36-39.

Reconfiguration-awareness

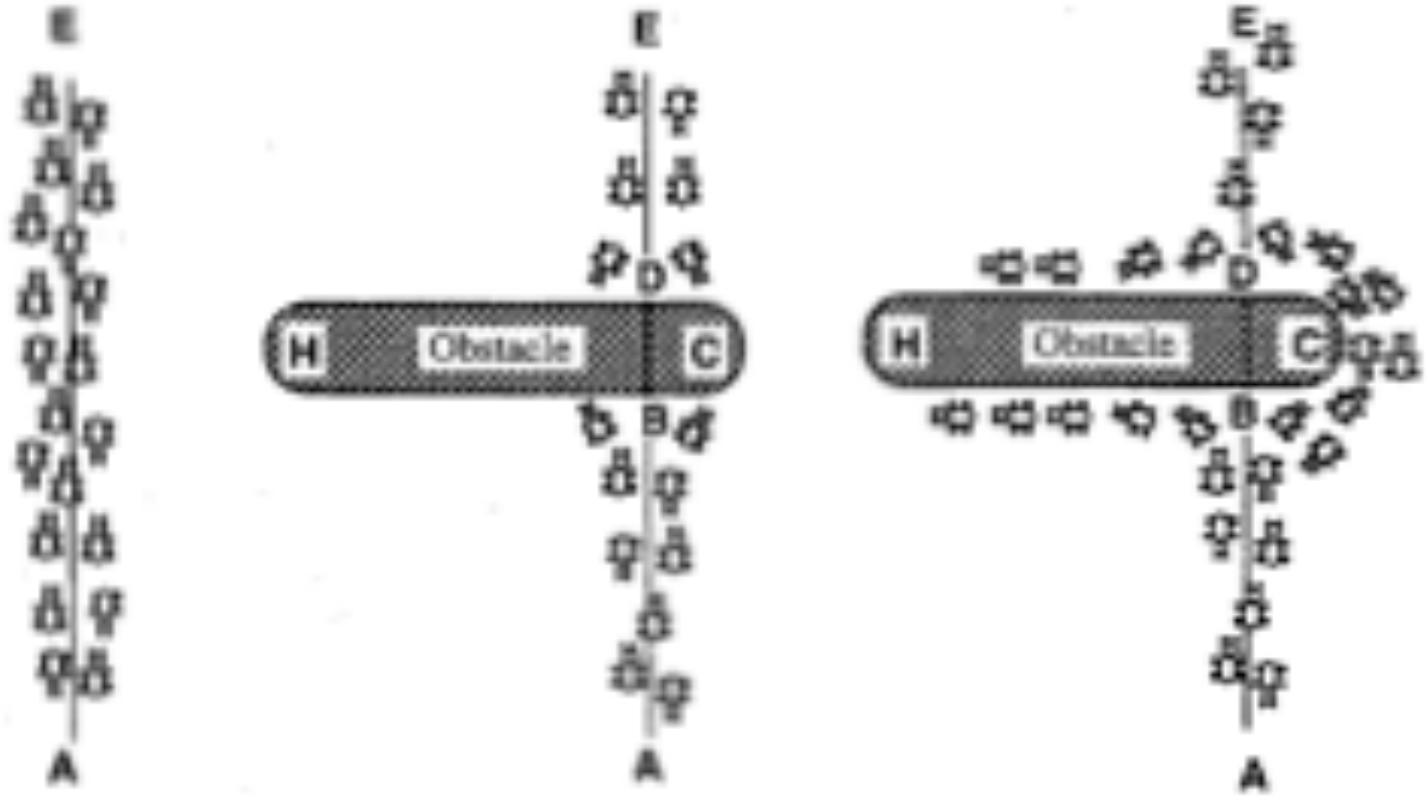
Reconfiguration-related specifics:

- at the **mapping** level
  - notion of **resource re-usage**
  - assignment of multiple implementations to single regions must not violate global resource usage constraint, so as to generate **only feasible solutions**
  - in the **heuristics**, do not allow to use too many logic resources too early in the mapping process to allow for potentially late reconfigurations to occur
- at the **scheduling** level
  - naïve first come first serve, to **estimate the execution time** of a given mapping (trading off accuracy for algorithm execution time)
  - take into account the **time** required **to reconfigure** a module into an other
  - take into account communication tasks also between successive reconfigurations

Premise: what is ACO?

Ant Colony Optimization

Metaheuristic Optimization Algorithm based on the ant colony metaphor



Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE transaction on Systems, MCn, and cybernetics-Part B: Cybernetics*, *26*(1), 29–41. doi:10.1109/3477.484436

Mapping: details

NP-Hard problem[1]. We approached its solution with ACO.

```
forall the numGenerations do
    forall the numAnts do
        readyTasksSet ← tasksWithoutPredecessors()
        scheduledTasksSet = ∅
        while readyTasksSet ≠ ∅ do
            forall the T_i ∈ readyTasksSet do
                p_{T_i} ← localHeuristic(T_i)
            end
            chosenTask ← rouletteSelection(p_T)
            iSet ←
            implementationsOfTask(chosenTask)
            forall the I_i ∈ iSet do
                pSet ←
                processorsPerImplementation(I_i)
                forall the P_i ∈ pSet do
                    p_{I_i,P_i} ← localHeuristic(I_i, P_i)
                end
            end
            chosenI, chosenP ← rouletteSelection(p_{I,P})
            mappingTrace.add(chosenT, chosenP, chosenI)
            readyTasksSet ← resolveDependencies()
        end
        ant.metrics ← computeMetrics(currentAnt)
        ant.objective ← computeObjective(metrics)
        thisGenerationSolutions.add(ant)
    end
    bestAnts ← selectBest(thisGenerationSolutions)
    updateGlobalPheromones(bestAnts)
end
return selectBest(bestAnts)
```

Local search

Choice of task

Choice of processor and implementation

Scheduling - Execution time

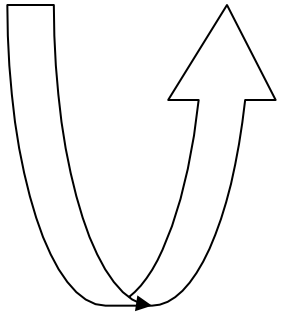
Evaluation of solution

Evolution of next generation

Global search

HLA produces estimates of the requirements of the implementations

Iterative (K generations)



[1] Merkle, D., Middendorf, M., & Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, *6*(4), 333–346. doi:10.1109/TEVC.2002.802450

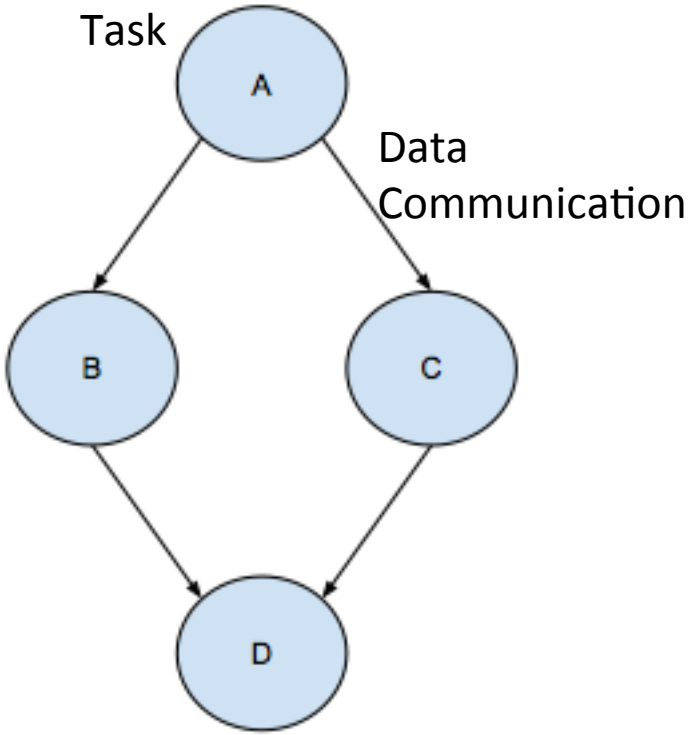
Mapping: heuristics

- Selection of task
  - Assign a **high value** to **lower mobility**, **lower average runtime**
- Selection of processor and implementation
  - Given an implementation
    - If software, assign a **high value** to this choice if
      - the processor with **least average assigned mobility** (averaged over previously assigned tasks)
    - If hardware, choose
      - If IP core, assign a **high value** to this choice if
        - » the implementation implements a **lot of tasks** (w.r.t. others)
        - » **average assigned mobility** is low
      - If FPGA, assign a **high value** to regions
        - » that are assigned a **low average mobility** (w.r.t. others)
        - » whose **increase in area consumption** after the association of this implementation to that region **is limited** w.r.t. the advancement of the mapping phase
        - » assign **0** to those choices that lead to **area constraint violation**

Scheduling / idea

Application's Taskgraph

(a.k.a. the workload, the application)

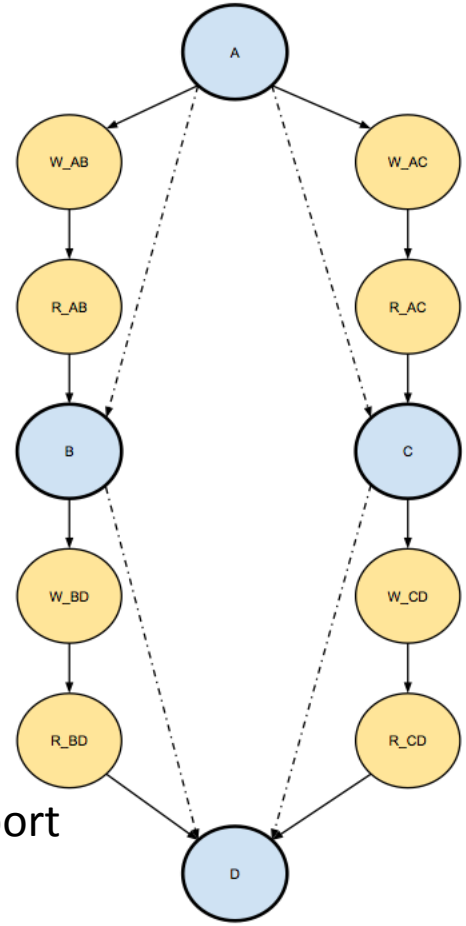


Scheduling / idea

Split communications tasks

Data read task

Data write task

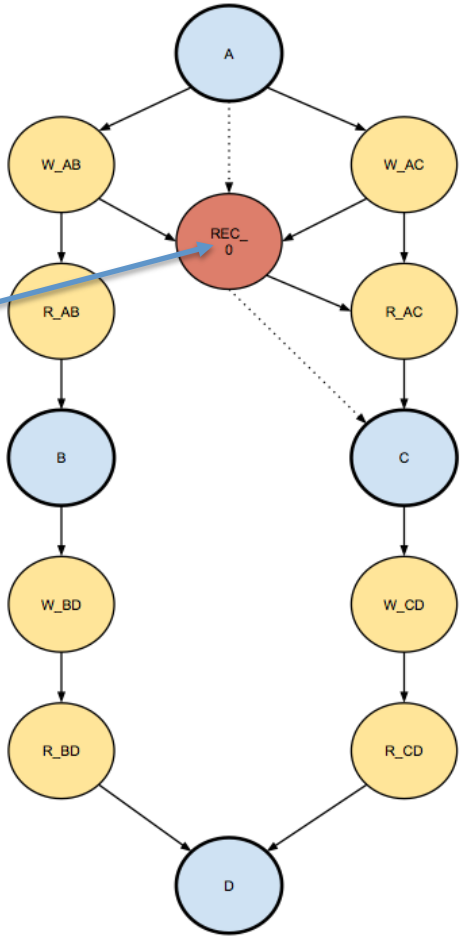


Implemented in any custom way – we support two communication architectures

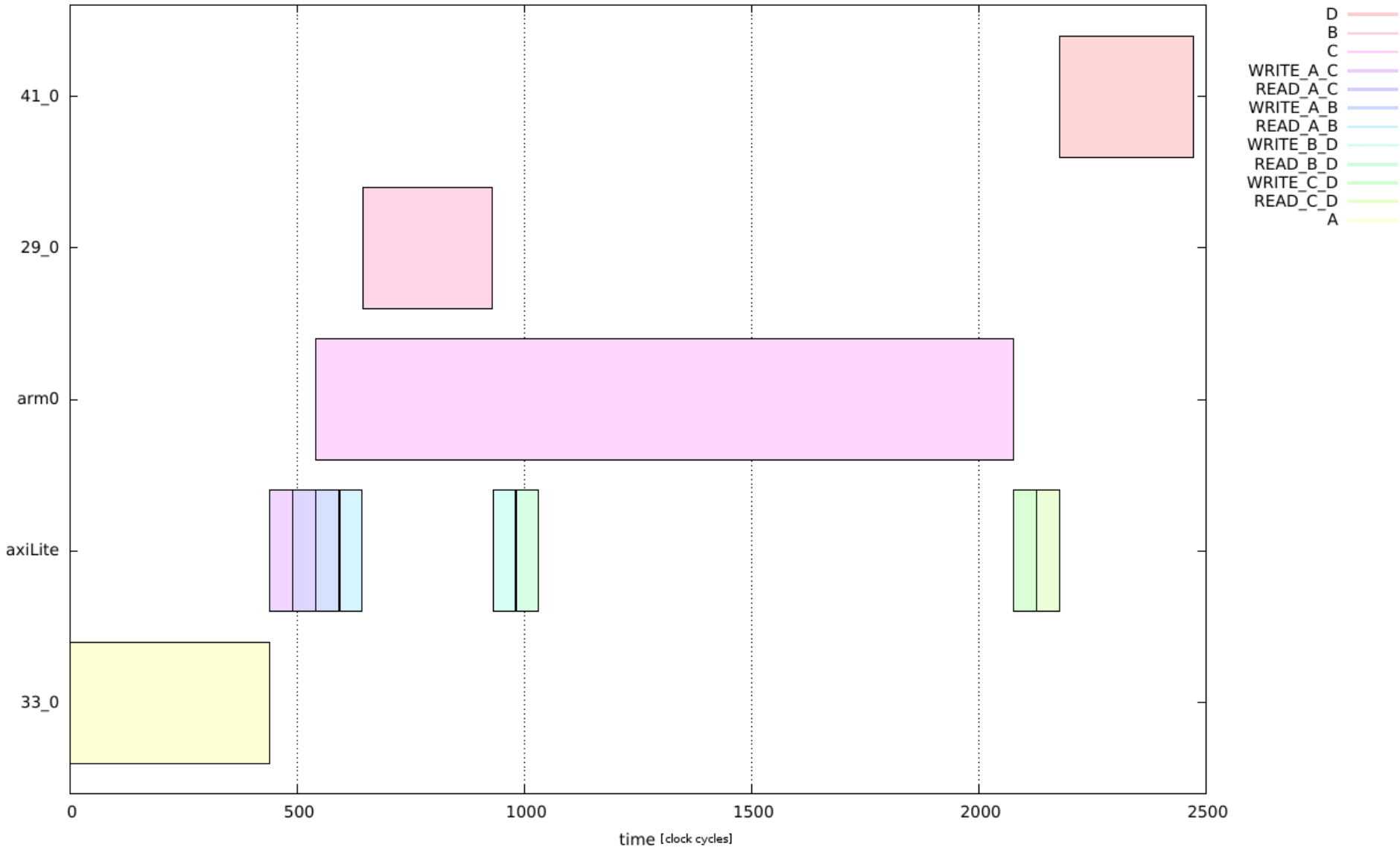
Scheduling / idea

Introduction of Reconfiguration task

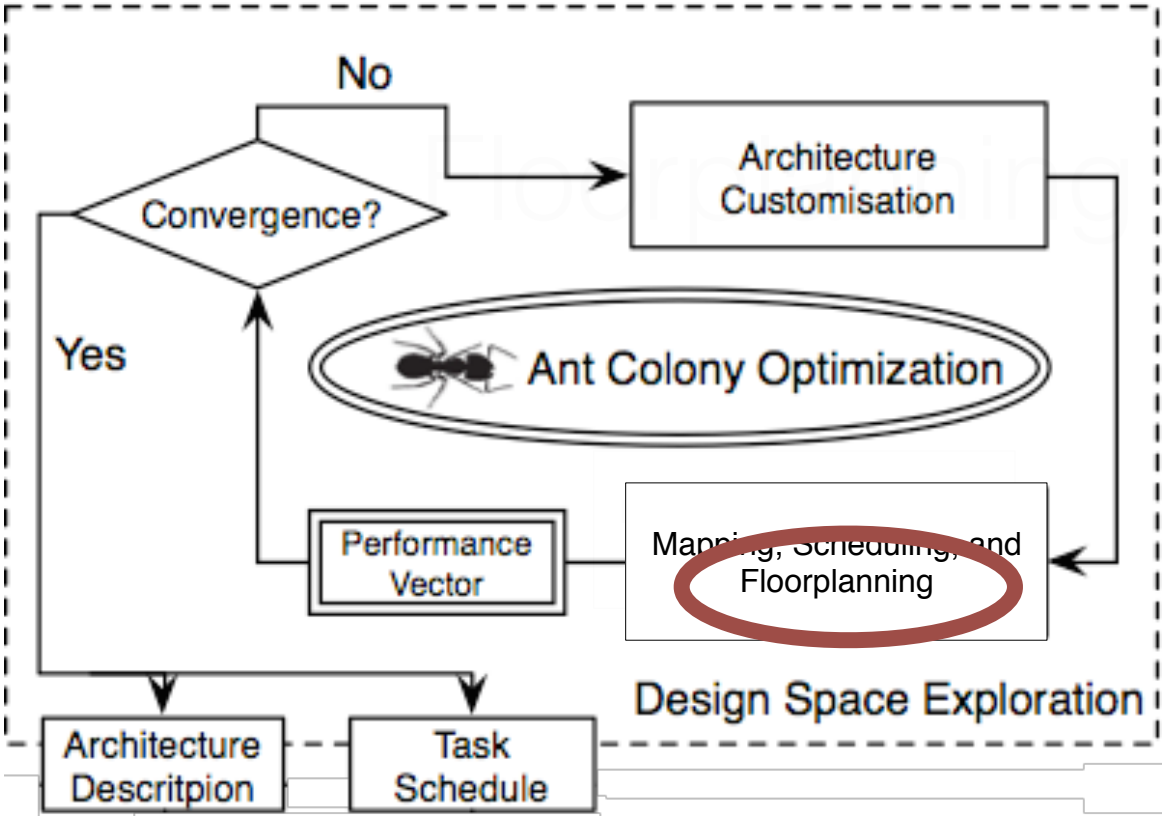
Correctness maintained



Example Gantt Output



Floorplanning / idea



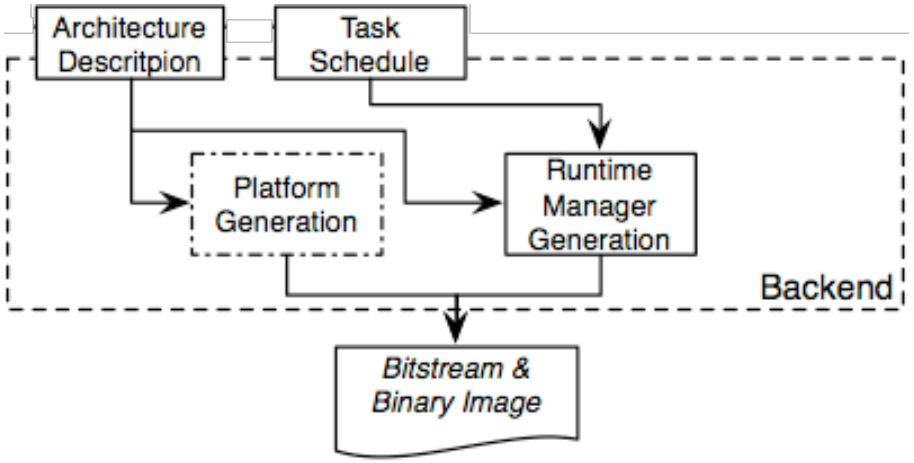
Goal: to automatically find reconfigurable regions' coordinates in FPGA

Floorplanning / idea

- Implemented as a Mixed Integer Linear Program (MILP)[1]
- Automatically computes reconfigurable regions' geometric bounds
  - Required by the PlanAhead flow
- Specifically aimed at reconfigurable systems employing Partial and Dynamic Reconfiguration
- Feedback to DSE: whether the computed solution is floorplannable
- More on this in the next talk

Backend

- Backend generates the runtime systems
  - Scheduler
  - Reconfiguration manager
- Backend generates the platform
  - *However, in a non-machine readable format*
  - *.mhs/.prj file compiled by hand, for now*
- Results are from actual execution on Zedboard (Zynq-7000 based)



- A little premise
- Problem statement & opportunities
- The FASTER approach
  - Motivation
  - Supported platforms and test cases
  - Methodology and framework overview
    - ACO-based mapper
    - Static scheduler and runtime manager
    - Code generation
- Experiments and results

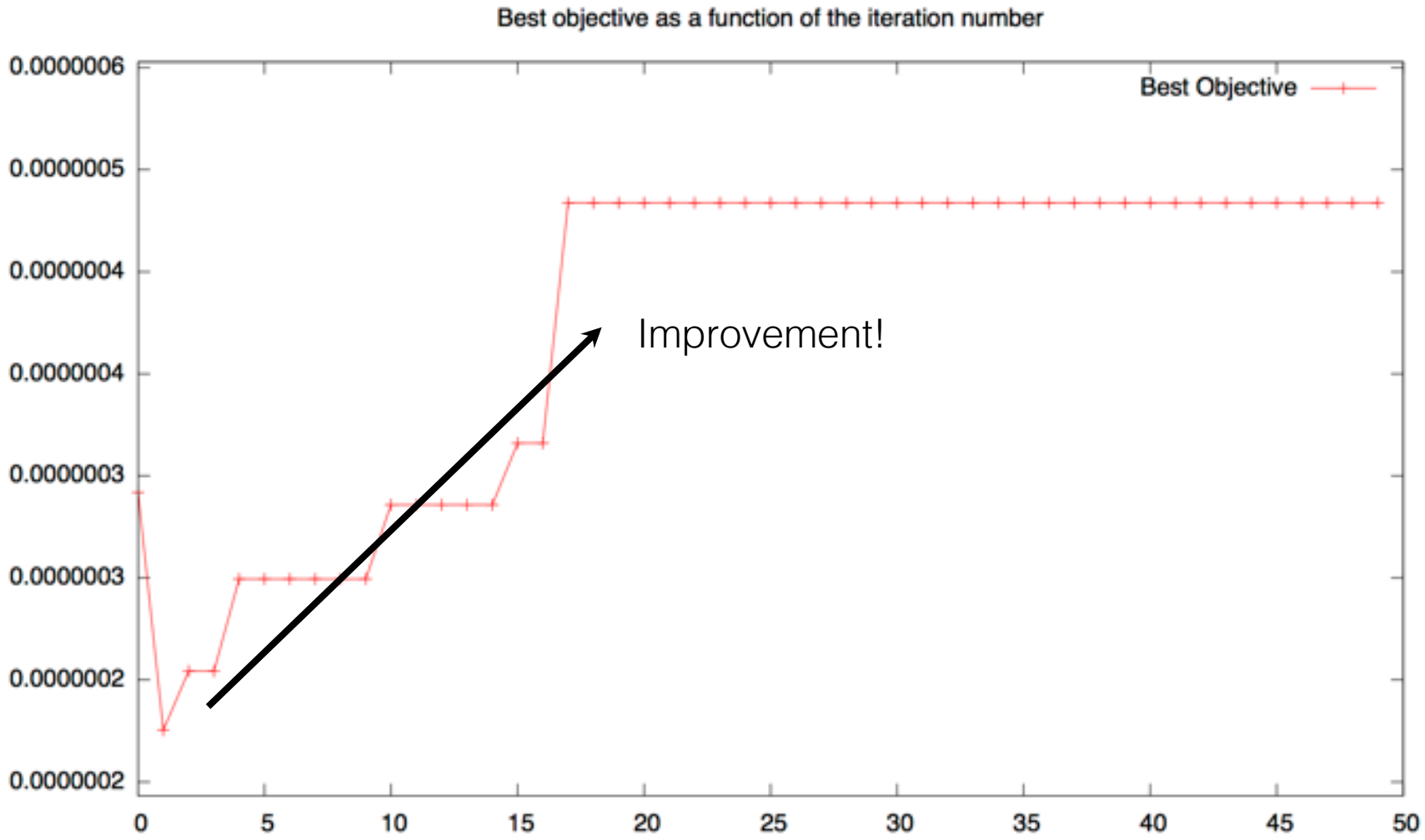
Experimental evaluation

Application - synthesized DAGs with 100 tasks, irregular topology (most complex case), many branches

Architecture - one generic processor, one reconfigurable area divided in up to 30 reconfigurable regions (architectural template which can be adequately modified)

Library - each task is assigned with one to four different available implementations (one software, 0-3 hardware with different area/performance tradeoffs)

Experimental evaluation



(Total execution time of the algorithm < 1h)

Experimental evaluation

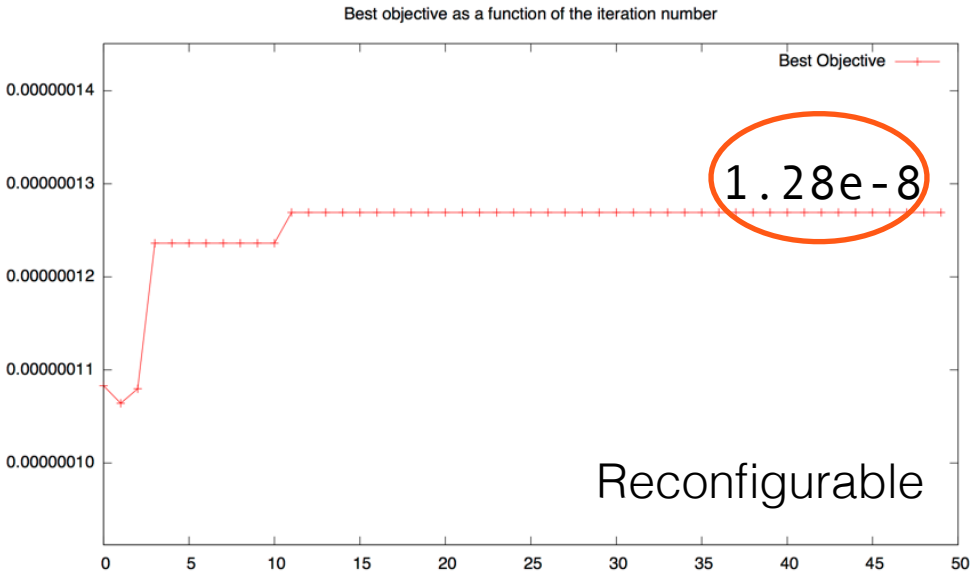
Static vs Reconfigurable design: objective function

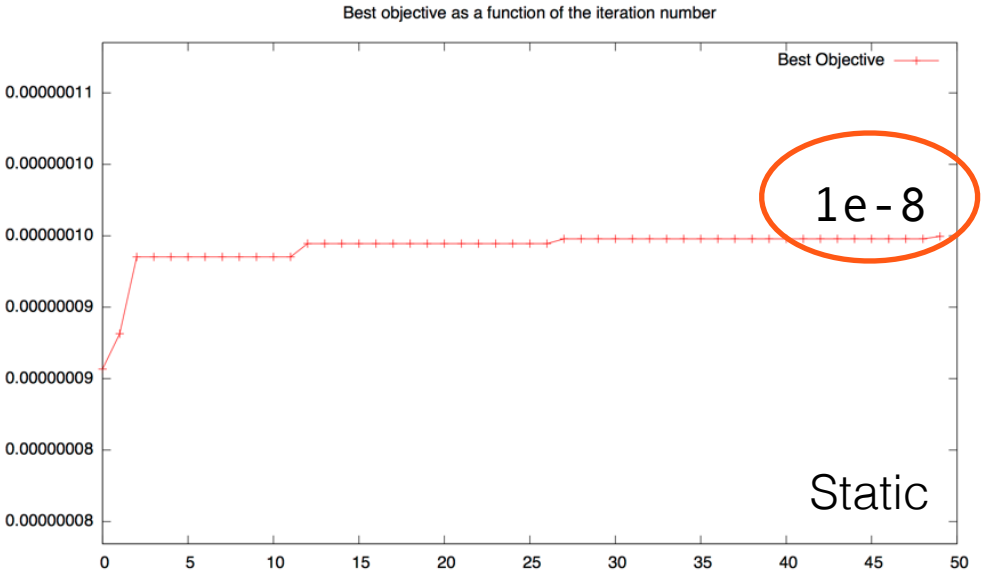
Static: allocate hardware tasks as long as you have area. Accelerate what you can, place the rest in software.

Reconfigurable: allocate hardware tasks as long as you have area. Accelerate what you can, **either reconfigure to keep accelerating in hardware** or place the rest in software.

Case study: 100.000 available LUTs, 100 tasks application input, 1-4 different implementations per task

Reconfigurable design performs **better** (the higher the objective function value, the better the design)



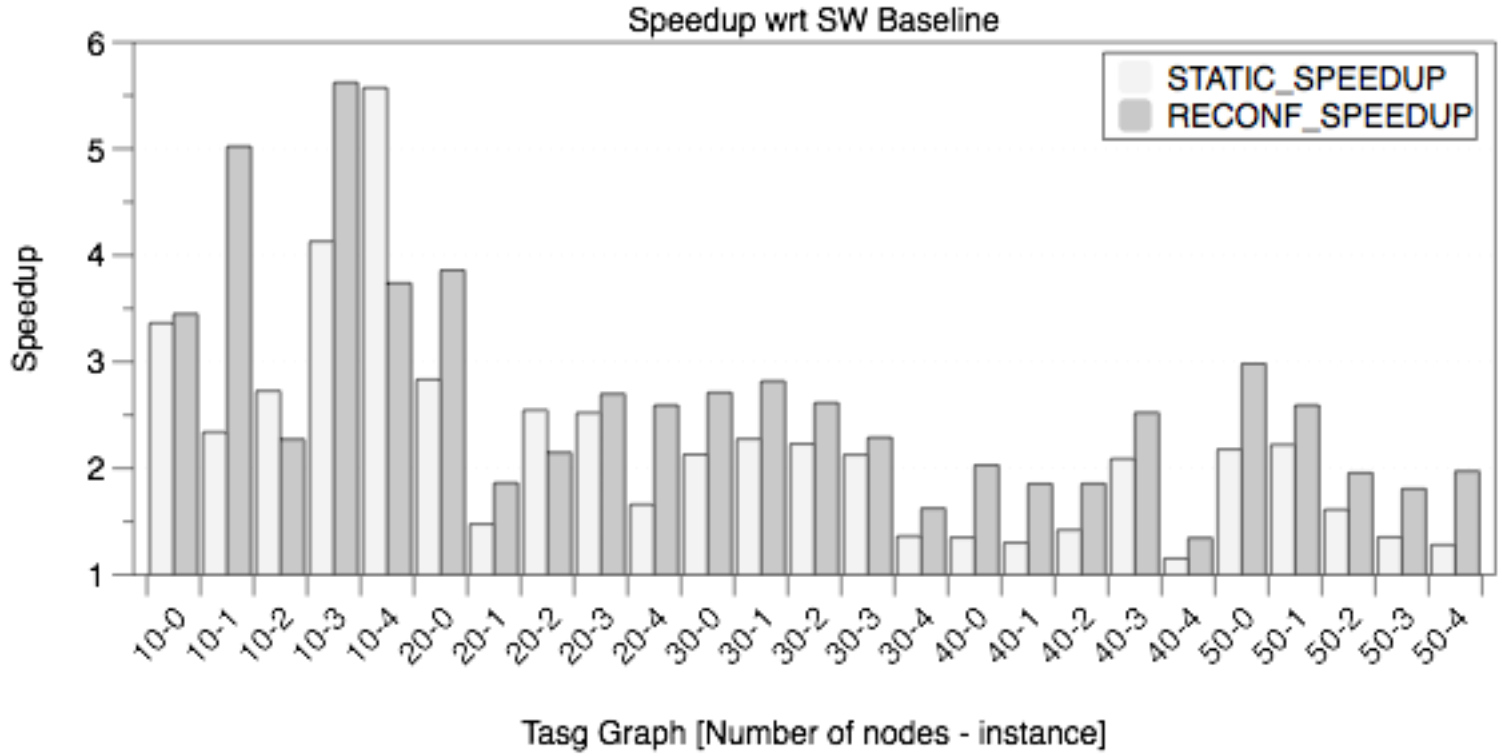


ARCHITECTURES COMPARISON

Light grey: architecture with static hardware accelerators + sw cores
Dark grey: architecture with reconfigurable hardware accelerators + sw cores
Baseline: Generic processor

➔ Reconfigurable architecture better exploits HW resources, on average

➔ Automatic time multiplexing of resources

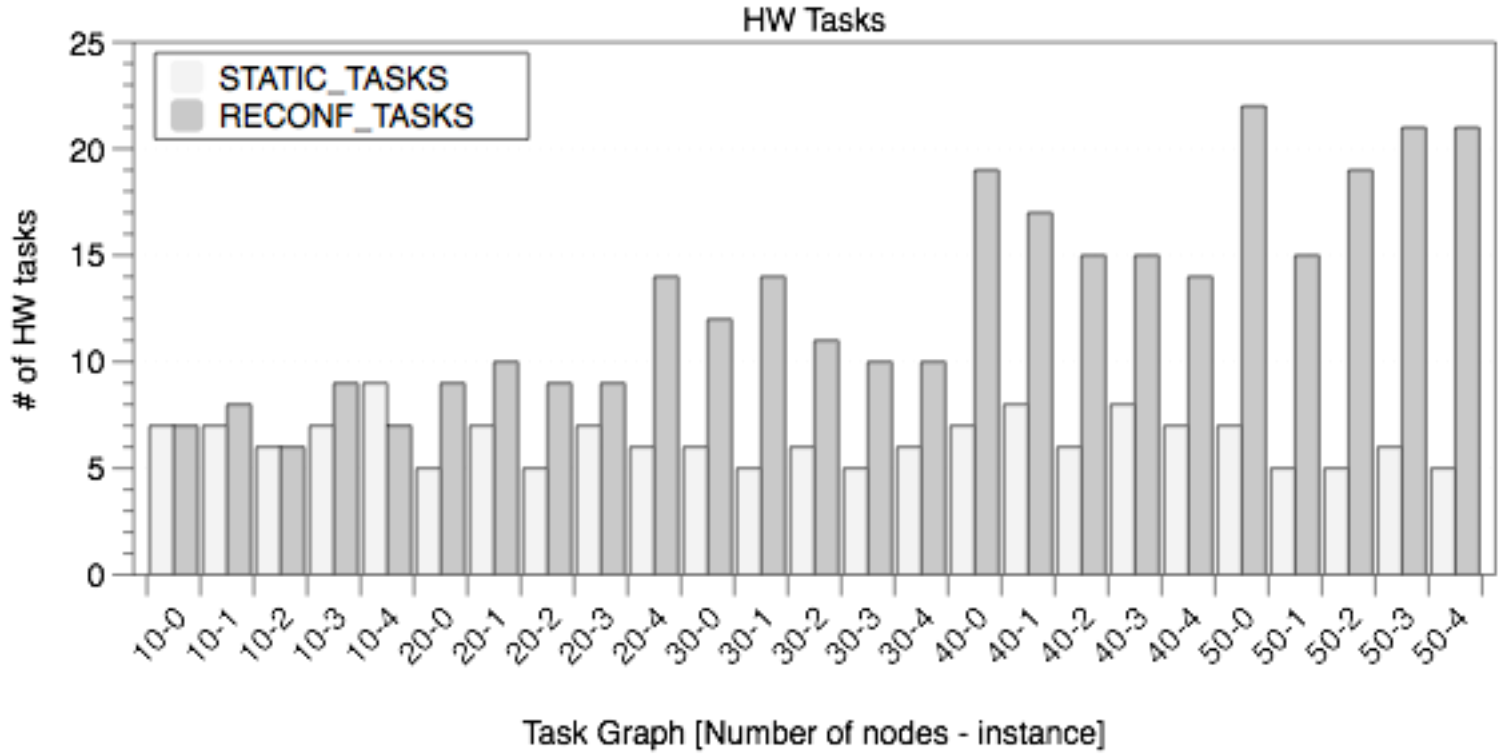


NUMBER OF TASKS ACCELERATED IN HARDWARE AT LEAST ONCE

Light grey: architecture with static hardware accelerators + sw cores
Dark grey: architecture with reconfigurable hardware accelerators + sw cores

➔ As expected, more tasks are run in hardware than in static case

➔ DSE automatically computes if and when to do so by keeping into account impact of reconfiguration and communication time

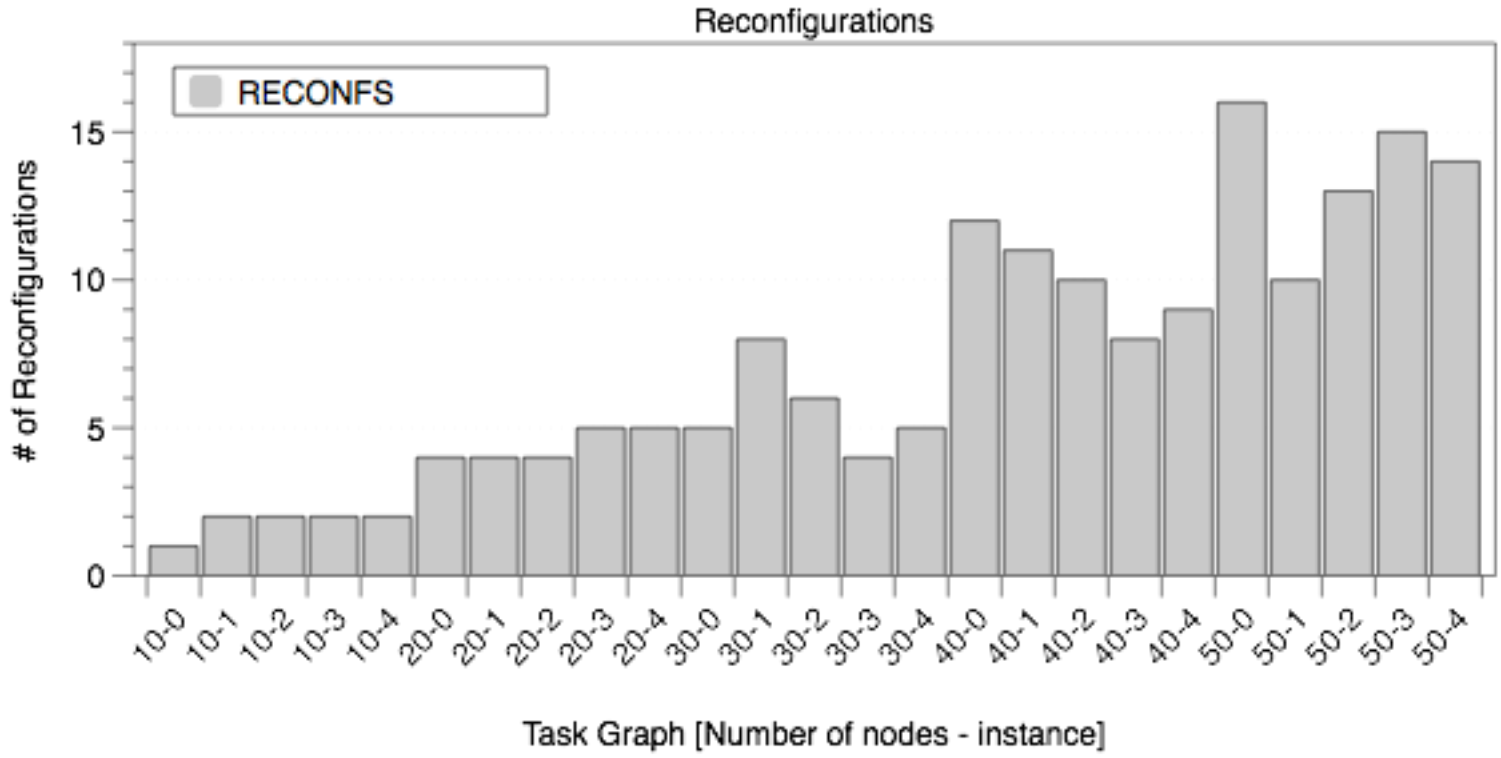


NUMBER OF RECONFIGURATIONS

Only for reconfigurable architecture: as the application grows in size, the number of reconfigurations increases as well.

➔ Efficient exploitation of hw resources by means of reusing (again, time multiplexing)

➔ As makespan showed, these reconfigurations are masked (i.e.: the acceleration they induce is larger than the cost of reconfiguration)

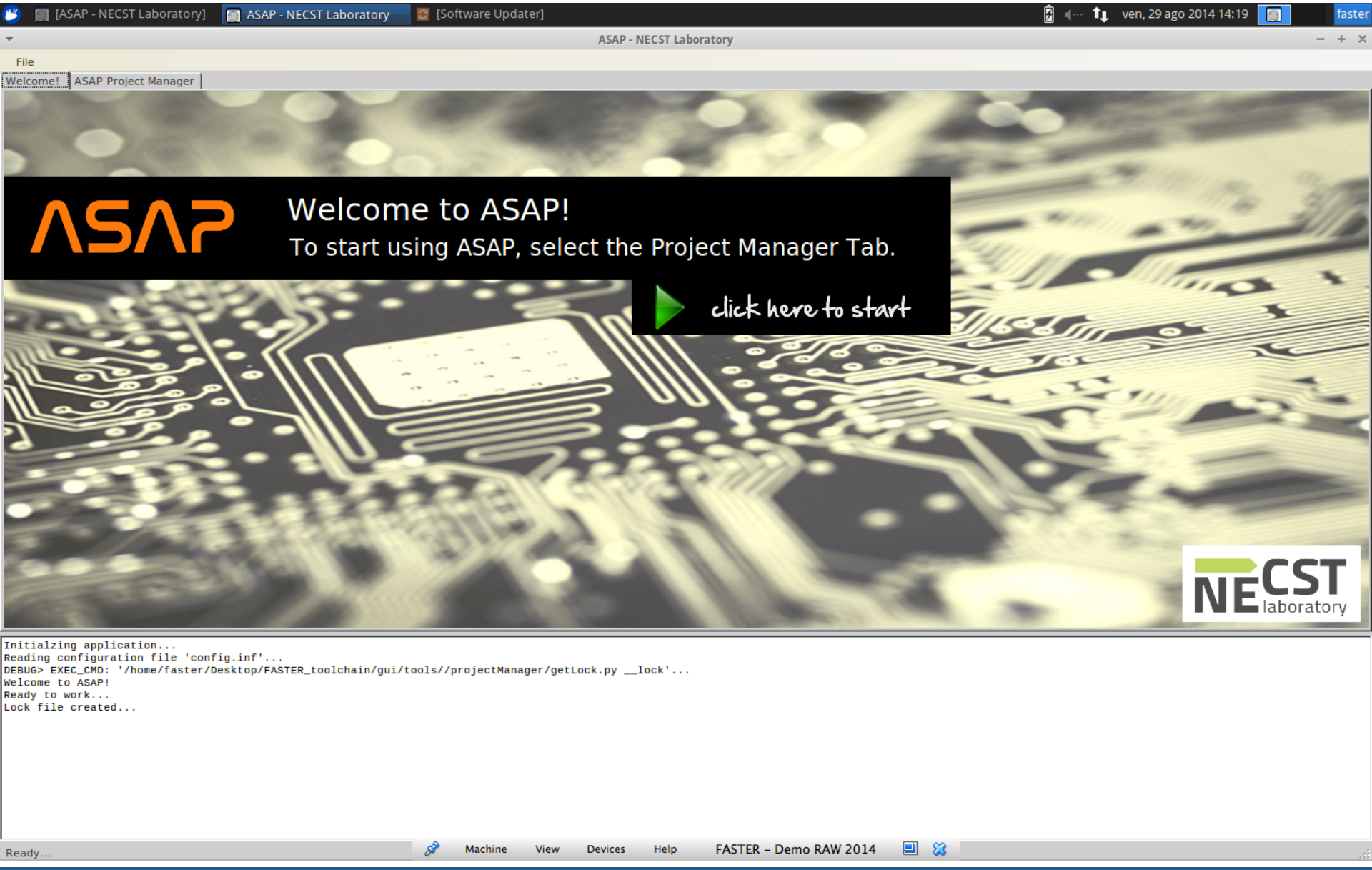


Conclusions

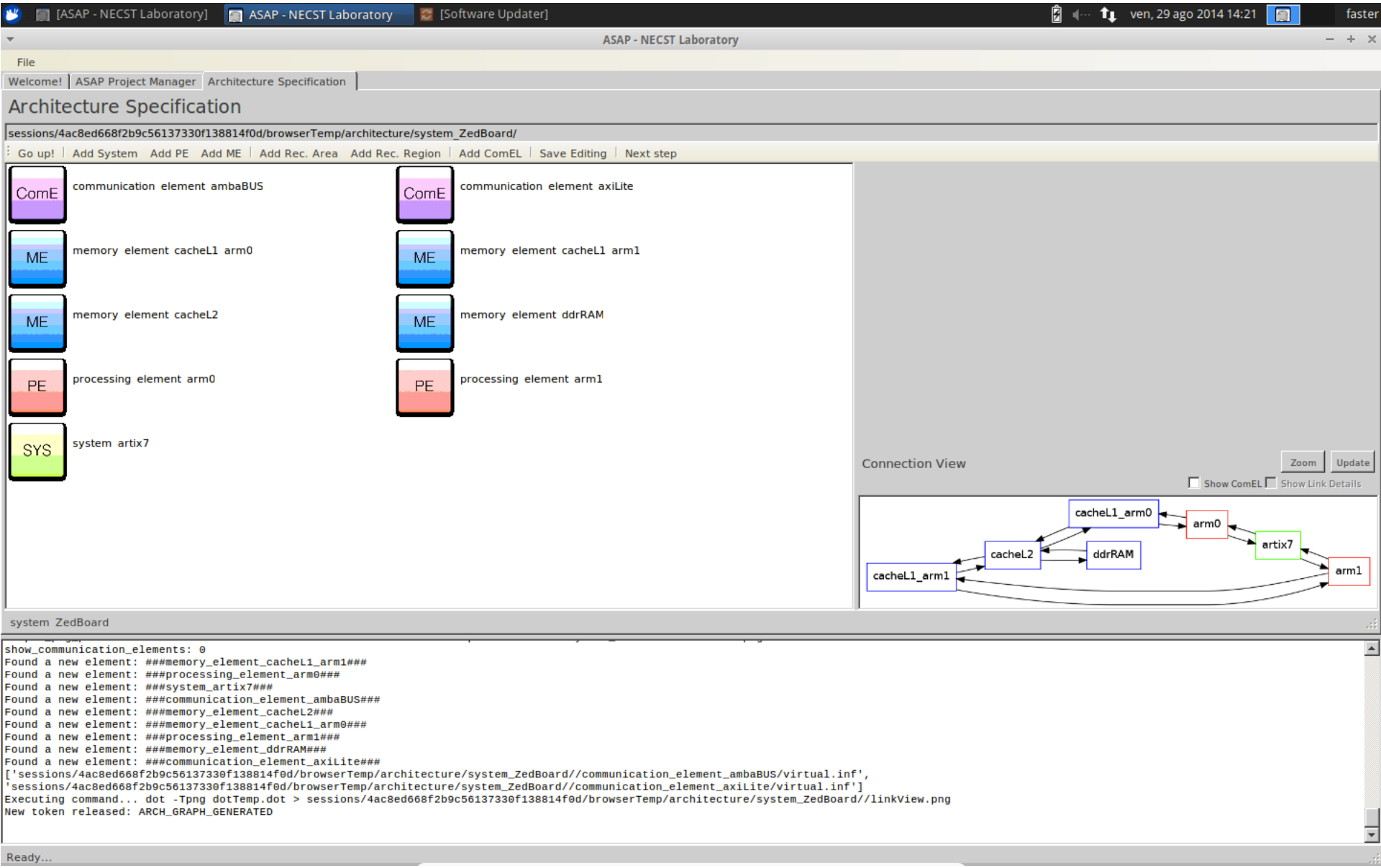
- We presented part of the toolchain FASTER, a fully-featured suite of tools for designing and implementing partially reconfigurable systems
- We demonstrated benefits of employing PDR on synthetic applications
- The system greatly enhances design productivity and early discovery of PDR employment benefits

GUI - DEMO

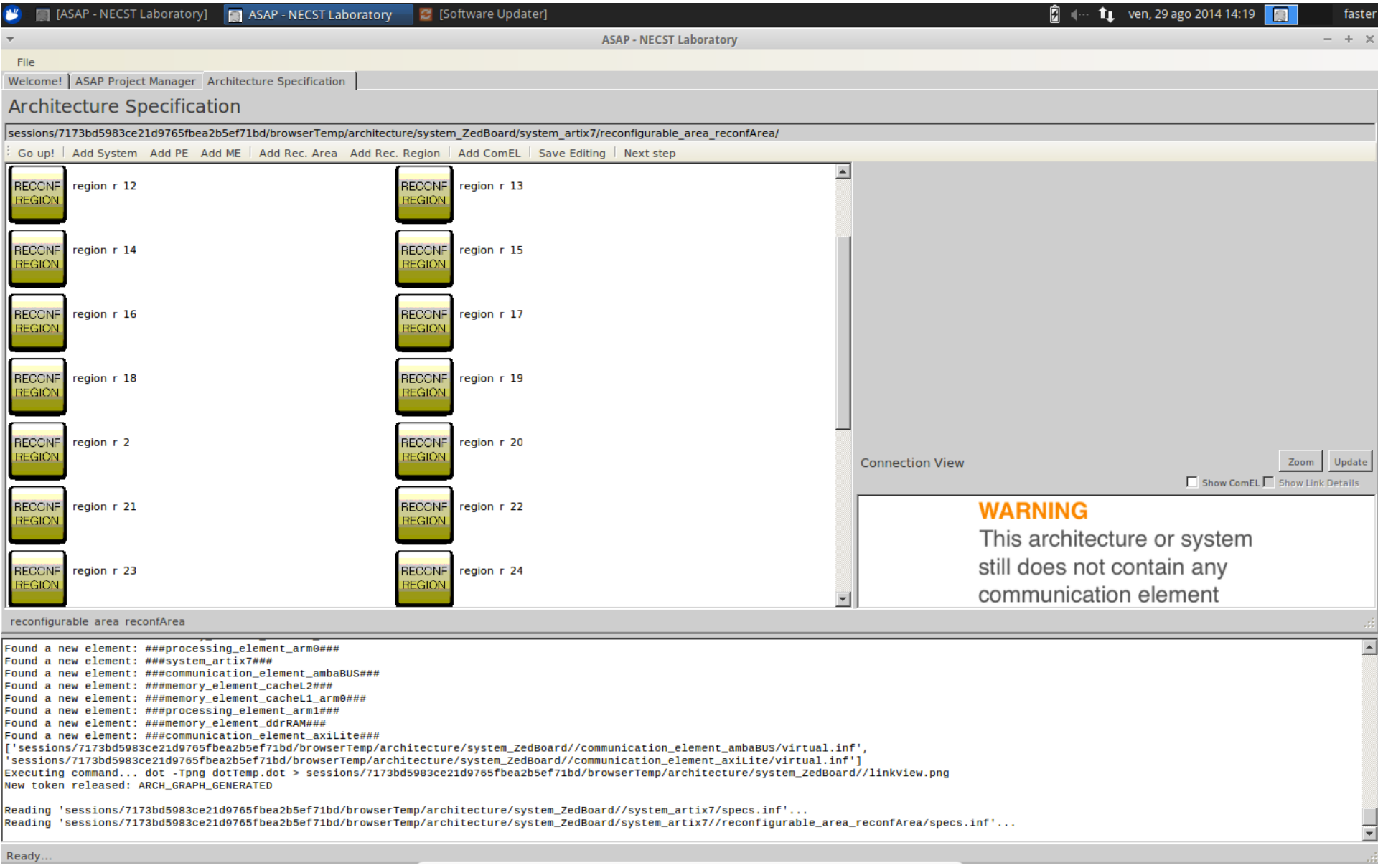
Initialization



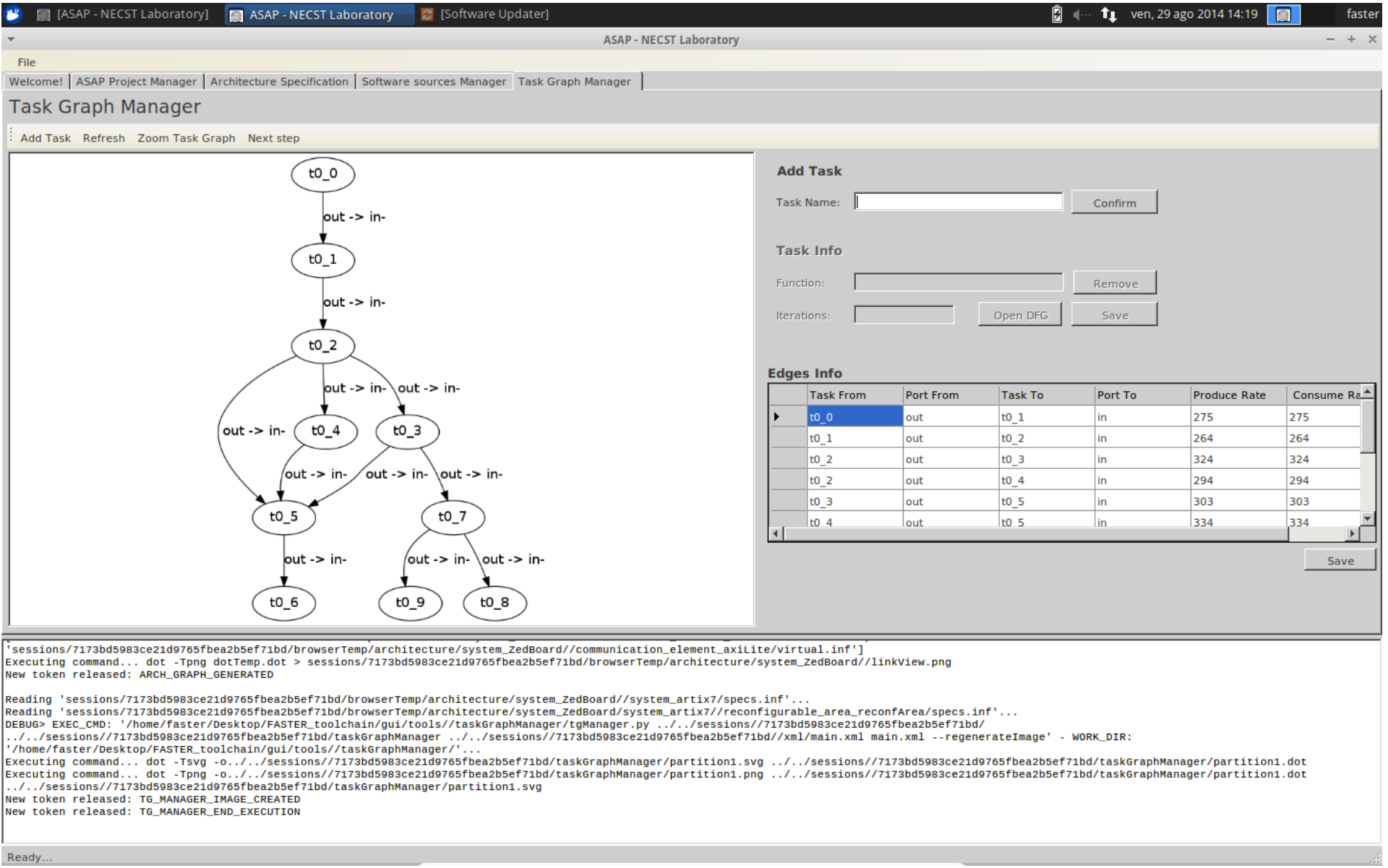
Zynq Architecture Model



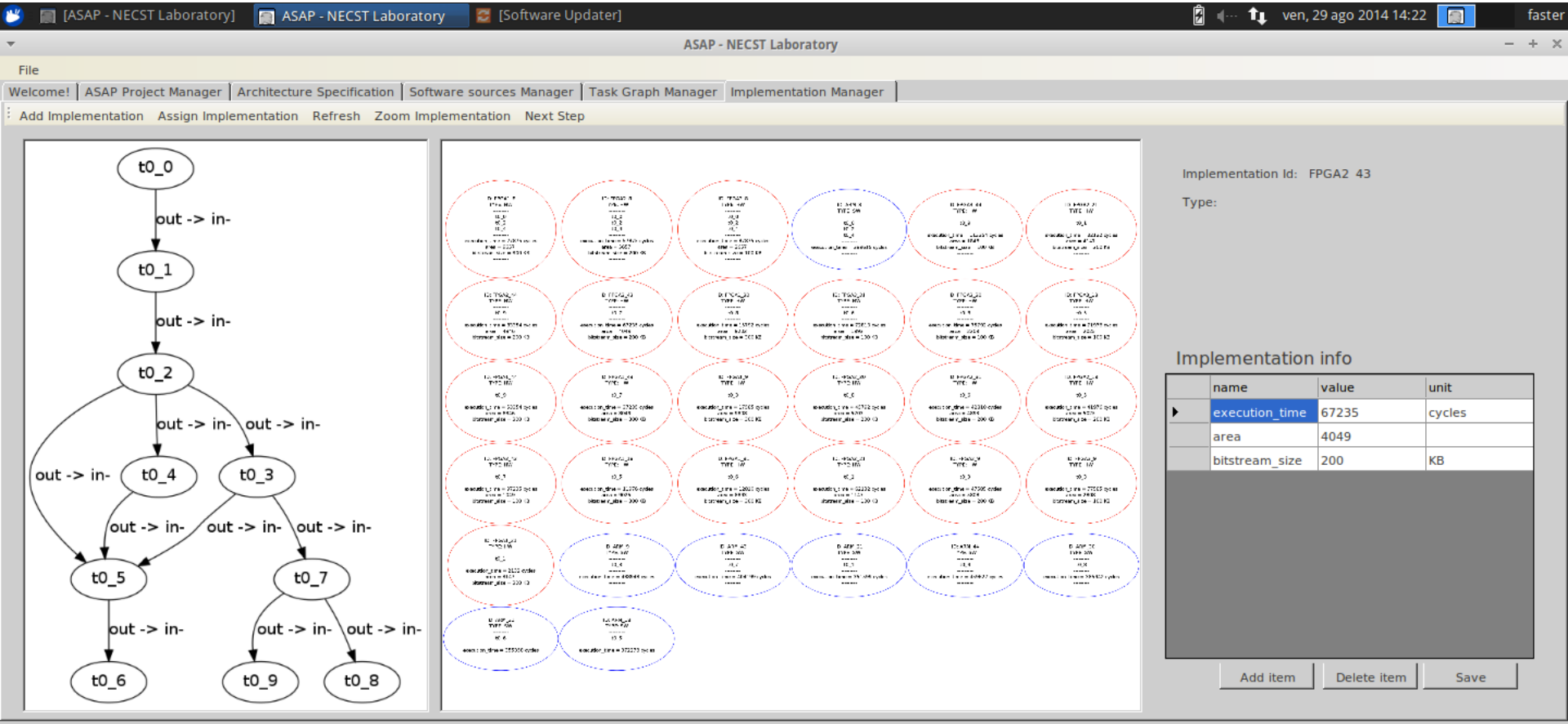
Architecture (reconfigurable area)



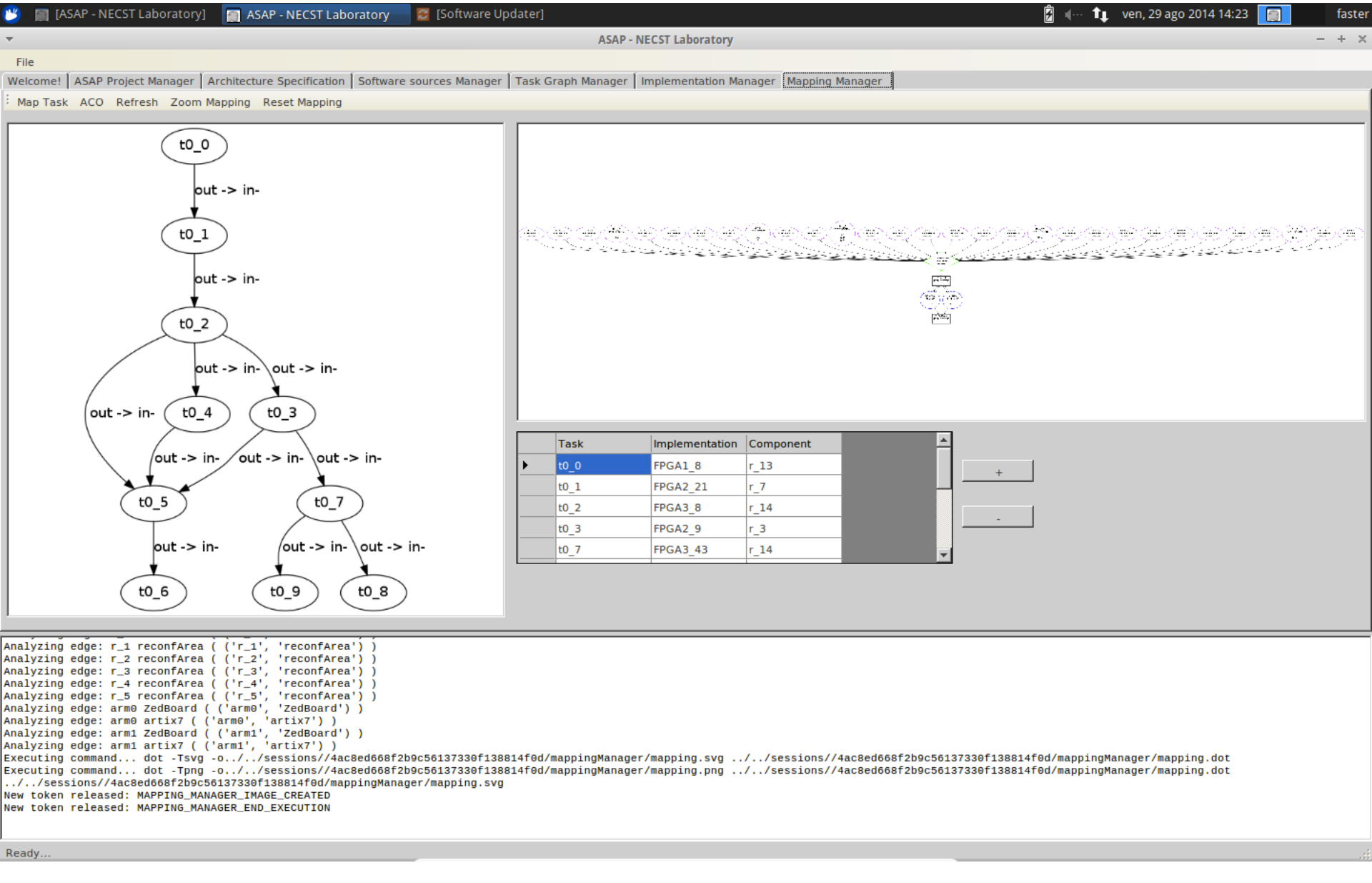
Taskgraph


[ASAP - NECST Laboratory]
ASAP - NECST Laboratory
[Software Updater]
ven, 29 ago 2014 14:19
faster
ASAP - NECST Laboratory
File
Welcome!
ASAP Project Manager
Architecture Specification
Software sources Manager
Task Graph Manager
Task Graph Manager
Add Task
Refresh
Zoom Task Graph
Next step
t0_0
out -> in-
t0_1
out -> in-
t0_2
out -> in-
out -> in-
out -> in-
t0_4
t0_3
out -> in-
out -> in-
out -> in-
out -> in-
t0_5
t0_7
out -> in-
out -> in-
out -> in-
t0_6
t0_9
t0_8
Add Task
Task Name:
Confirm
Task Info
Function:
Remove
Iterations:
Open DFG
Save
Edges Info
Task From
Port From
Task To
Port To
Produce Rate
Consume Ra
t0_0 out t0_1 in 275 275
t0_1 out t0_2 in 264 264
t0_2 out t0_3 in 324 324
t0_2 out t0_4 in 294 294
t0_3 out t0_5 in 303 303
t0_4 out t0_5 in 334 334
Save
'sessions/7173bd5983ce21d9765fbea2b5ef71bd/browserTemp/architecture/system_ZedBoard//communication_element_axiLite/virtual.inf']
Executing command... dot -Tpng dotTemp.dot > sessions/7173bd5983ce21d9765fbea2b5ef71bd/browserTemp/architecture/system_ZedBoard//linkView.png
New token released: ARCH_GRAPH_GENERATED
Reading 'sessions/7173bd5983ce21d9765fbea2b5ef71bd/browserTemp/architecture/system_ZedBoard//system_artix7/specs.inf'...
Reading 'sessions/7173bd5983ce21d9765fbea2b5ef71bd/browserTemp/architecture/system_ZedBoard/system_artix7//reconfigurable_area_reconfArea/specs.inf'...
DEBUG> EXEC_CMD: '/home/faster/Desktop/FASTER_toolchain/gui/tools//taskGraphManager/tgManager.py ../../sessions//7173bd5983ce21d9765fbea2b5ef71bd/
../../sessions//7173bd5983ce21d9765fbea2b5ef71bd/taskGraphManager ../../sessions//7173bd5983ce21d9765fbea2b5ef71bd//xml/main.xml main.xml --regenerateImage' - WORK_DIR:
'/home/faster/Desktop/FASTER_toolchain/gui/tools//taskGraphManager/'...
Executing command... dot -Tsvg -o../../sessions//7173bd5983ce21d9765fbea2b5ef71bd/taskGraphManager/partition1.svg ../../sessions//7173bd5983ce21d9765fbea2b5ef71bd/taskGraphManager/partition1.dot
Executing command... dot -Tpng -o../../sessions//7173bd5983ce21d9765fbea2b5ef71bd/taskGraphManager/partition1.png ../../sessions//7173bd5983ce21d9765fbea2b5ef71bd/taskGraphManager/partition1.dot
../../sessions//7173bd5983ce21d9765fbea2b5ef71bd/taskGraphManager/partition1.svg
New token released: TG_MANAGER_IMAGE_CREATED
New token released: TG_MANAGER_END_EXECUTION
Ready...

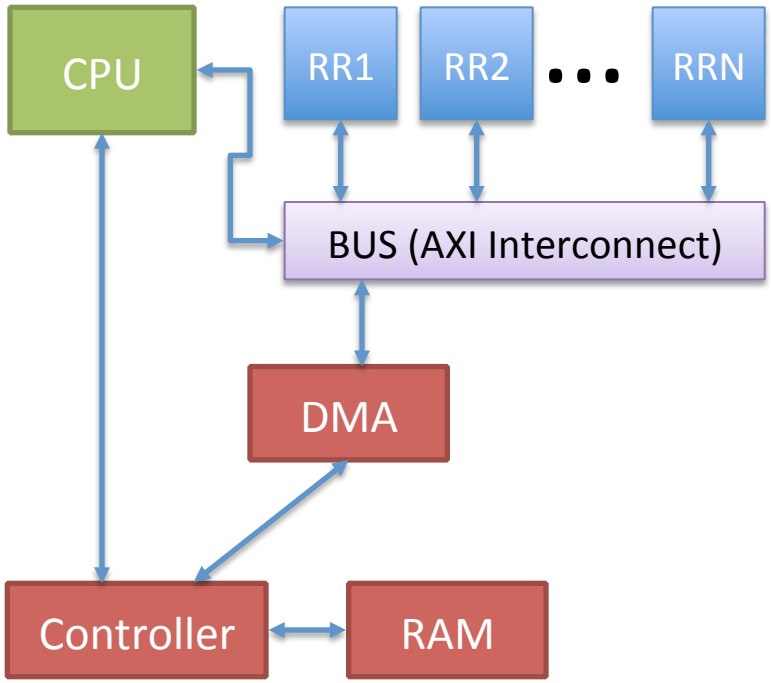
Library/Implementations



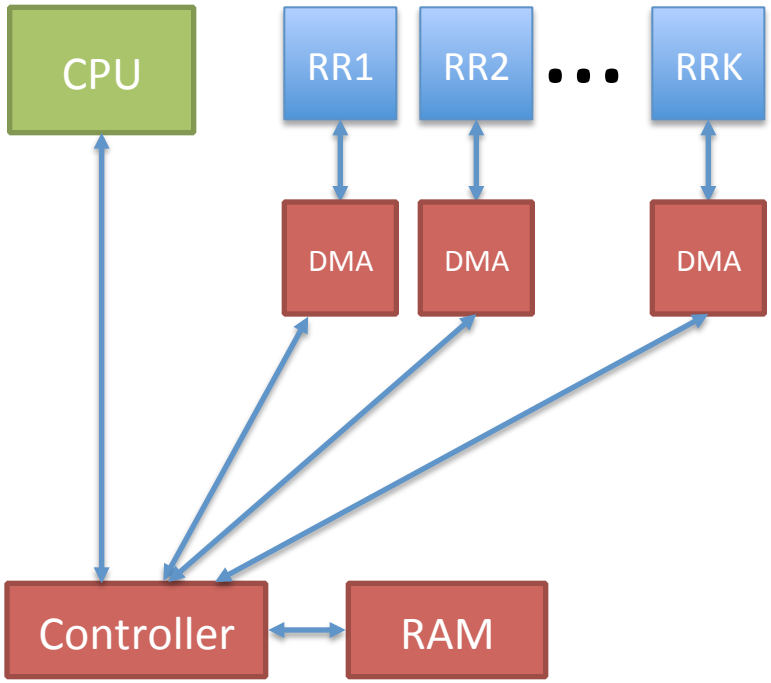
Mapped Applications



REFERENCE ARCHITECTURES



Up to K ports, single bus for N partitions



Up to K ports, dedicated DMA channel per reconfigurable region