北京大学高能效计算与应用中心
Center for Energy-efficient Computing and Applications

Cocoon: An Open-Source Infrastructure for Integrated EDA with Interoperability and Interactivity

Jiaxi Zhang[1], Tuo Dai[1], Zhengzheng Ma[1,2], Yibo Lin[1], Guojie Luo[1,2]

[1]Center for Energy-Efficient Computing and Applications, Peking University, Beijing, China

[2]Peng Cheng Laboratory, Shenzhen, China

Email: {zhangjiaxi, gluo}@pku.edu.cn

Background and Motivation

- Background
  - Modern IC design requires the joint efforts of EDA tool developers, system integrators and IC designers
  - “The EDA companies had grown from a lot of acquisitions so that’s what they had for sale: good point tools that were poorly integrated.” -- Paul McLellan in EDAgraffiti
- Motivation
  - Over-optimized point tools + poorly integrated flow ⇒ Moderately optimized customized tools + easy-to-customized integrated flow
  - The existing open source flows still need extended development to be available
    - OpenRoad Flow [1], DATC Robust Design Flow [2]
  - Similar attempts
    - EDAlize [3], Hammer [4]

[1] T. Ajayi, et al., “Toward an open-source digital flow: First learnings from the OpenROAD project,” DAC 2019.
[2] J. Jung, et al., “DATC RDF: Robust design flow database,” ICCAD 2017.
[3] Edalize: https://github.com/olofk/edalize
[4] Hammer: https://github.com/ucb-bar/hammer

Integrated EDA

- What is Integrated EDA?
  - Expanded from Electronic CAD Framework [5]
  - A system composed of EDA point tools, designs, and interfaces
  - Users: EDA researchers, tool developers, and IC designers
- Ideal Characteristics of Integrated EDA
  - Interoperability (Within the EDA flow)
    - The ability of two or more point tools to exchange design information
    - Support the mixing of tools from different vendors of open source community
  - Interactivity (Beyond the EDA flow)
    - A unified user interface, supporting users to flexibly select and deploy different point tools
    - A unified and abstract programming interface, supporting automated design methodology research and flow tuning

[5] Barnes T J, Harrison D, Newton A R, et al. Electronic CAD frameworks[M]. Springer Science & Business Media, 2012.

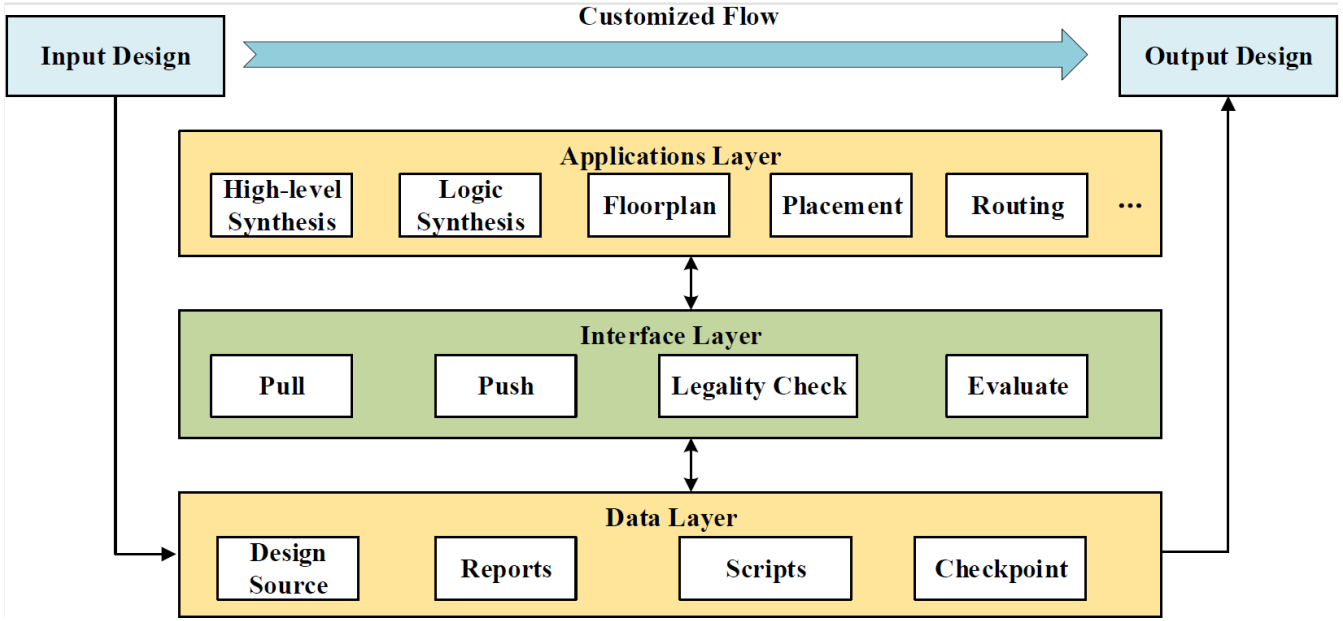
Integrated EDA

- Possible Solutions of Integrated EDA
  - Open standards: Liberty, LEF/DEF, SDC, etc.
  - Cross tool API: Higher level of API abstraction than TCL
  - Steps Abstraction: logic synthesis, place, cts, route, etc. Legality check are need
  - EDA brokers: Tools or tool parameters recommendation (Human experts or AI)
  - Model-based DSL: Domain specific language for design flows

Approaches in Inter-connected Cloud	**Corresponding approaches in Integrated EDA**	**Available in existing flows?**
Open Standards/Protocols	Open Standards	YES
Cross-platform APIs	Cross-tool APIs	NO
Layers Abstraction	Steps Abstraction	YES
Cloud Brokers/Agents	EDA Brokers/Agents	NO
Model-based DSL	Flow-based DSL	NO

Cocoon: Architecture

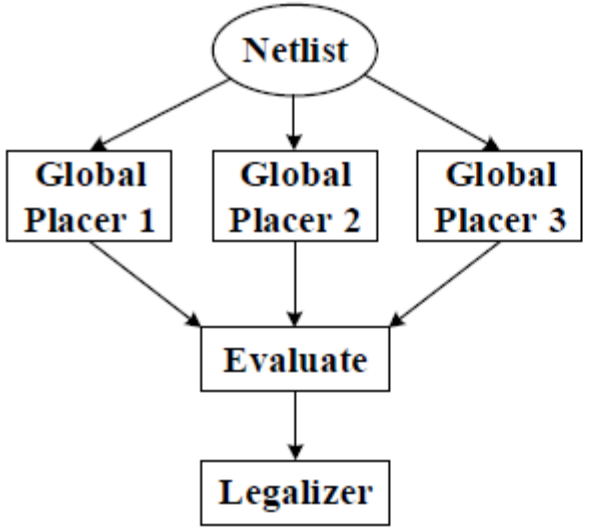
- Key Features
  - Cross-tool APIs
    - Applications layer, interfaces layer, data layer
    - Checkpoint design (including history running scripts)
    - Fast designs and reports extraction
  - EDA Brokers
    - Customized flow and Legality Check Mechanism
    - Learning-based flow tuning and optimization



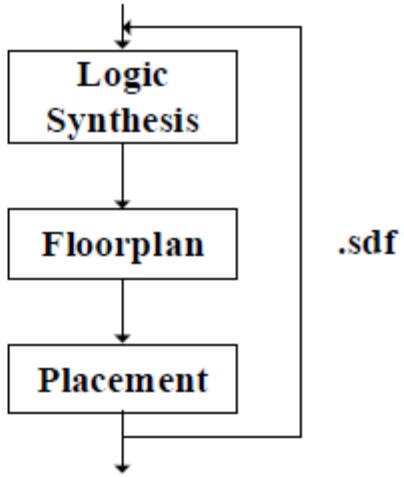
Repo: https://github.com/pku-dasys/cocoon

Cocoon: Customized Flow

- Customized Flow Definition
  - Branching flow
    - Scenarios1: Compare the better one and do next steps
    - Scenarios2: Partition the design and do each part parallel
  - Iterating flow
    - Scenarios1: Bad QoR
    - Scenarios2: Back annotation (physical-aware high level synthesis and logic synthesis)



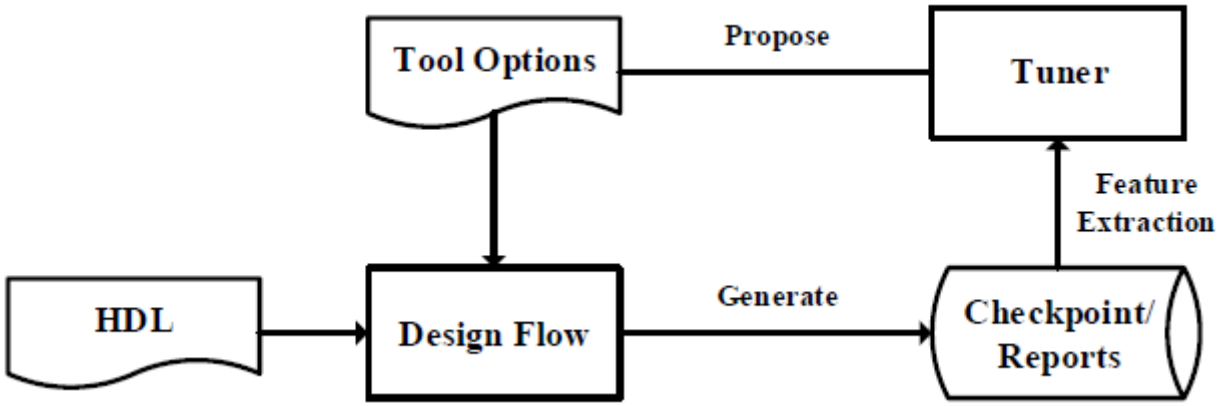
(a) Branching flow.



(b) Iterating flow.

Cocoon: Customized Flow

- Flow Tuning
  - EDA tools provide numerous options and parameters that can drastically impact design quality
  - Design space exploration
  - Method
    - Search strategies (SA, GA)
    - Parallel computing
    - Learning-based methods



Implementation and Evaluation

- Implementation
  - Python 3
  - Applications layer are implemented as classes
    - Each class will wrap the implementation of a tool, including member functions like parameters setting and scripts generation
  - Data layer is implemented as a database with a fixed directory
  - Interfaces layer are implemented as global functions
- Evaluation
  - Branching flow
    - Demo code
  - Flow tuning
    - Baseline method: Bayesian Optimization for black-box optimization

```
def flow(self):

    app_synth = []
    app_synth.append(("YosysSynth", "to_synth", "Timing"))
    app_synth.append(("GenusSynth", "to_synth", "Timing"))
    self.apps.append(app_synth)

    app_floorplan = [("InnovusFloorplan", "to_floorplan")]
    self.apps.append(app_floorplan)
    self.params_fp.append(("r","1.0 0.7 0.0 0.0 0.0 0.0"))

    app_pdn = [("InnovusPDN", "to_pdn")]
    self.apps.append(app_pdn)

    app_place = [("InnovusPlace", "to_place")]
    self.apps.append(app_place)

    app_cts = [("InnovusCTS", "to_cts")]
    self.apps.append(app_cts)

    app_route = [("InnovusRoute", "to_route")]
    self.apps.append(app_route)
```

Future work

- Beautify the UI for users
- Integrate more open source tools
- Implement legality check
- Cloud Infrastructure
  - Distributed checkpoint design
  - Computation graph scheduling of applications