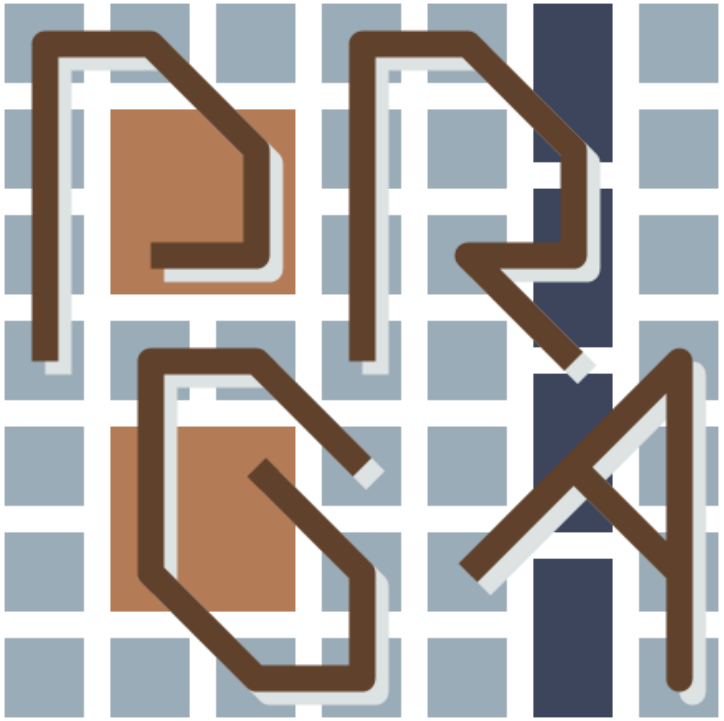


PRGA:

An Open-source Framework for Building and Using Custom FPGAs

Ang Li, David Wentzlaff

Princeton University

PRINCETON UNIVERSITY

github.com/PrincetonUniversity/prga

The Era of Open-Source Hardware

NVDLA **MIAOW**

OpenPiton

RISC-V®

OpenSPARC™

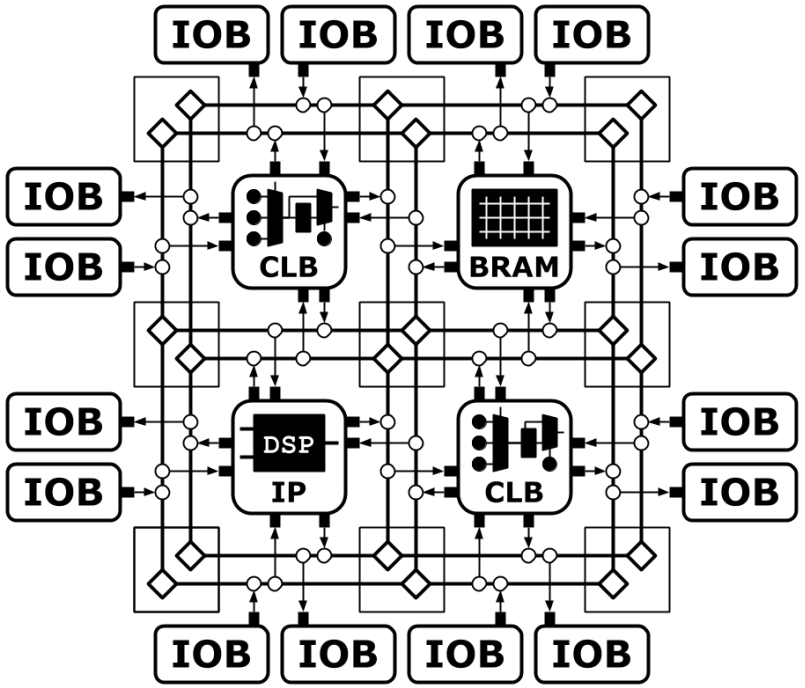
OpenRISC **PULP Ariane**

PRINCETON UNIVERSITY

github.com/PrincetonUniversity/prga

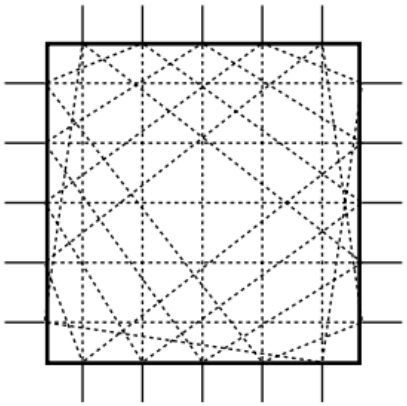
Princeton Reconfigurable Gate Array (PRGA)

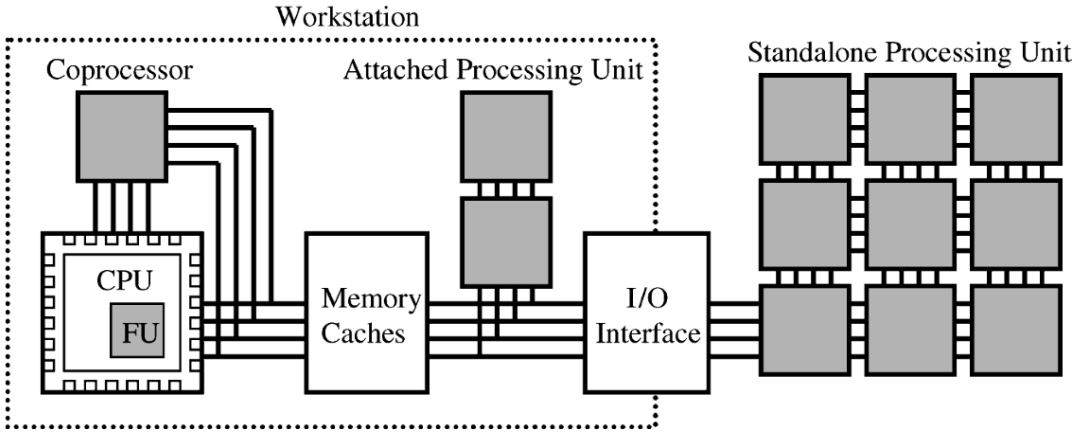
- Open-source FPGA generator
- Highly customizable, scalable, extensible, modularized
- Scripts for open-source CAD tools (Yosys, VPR, etc.)



Enabled Applications

- FPGA architecture exploration
- Embedded FPGA
- Specialized/Domain-specific FPGA





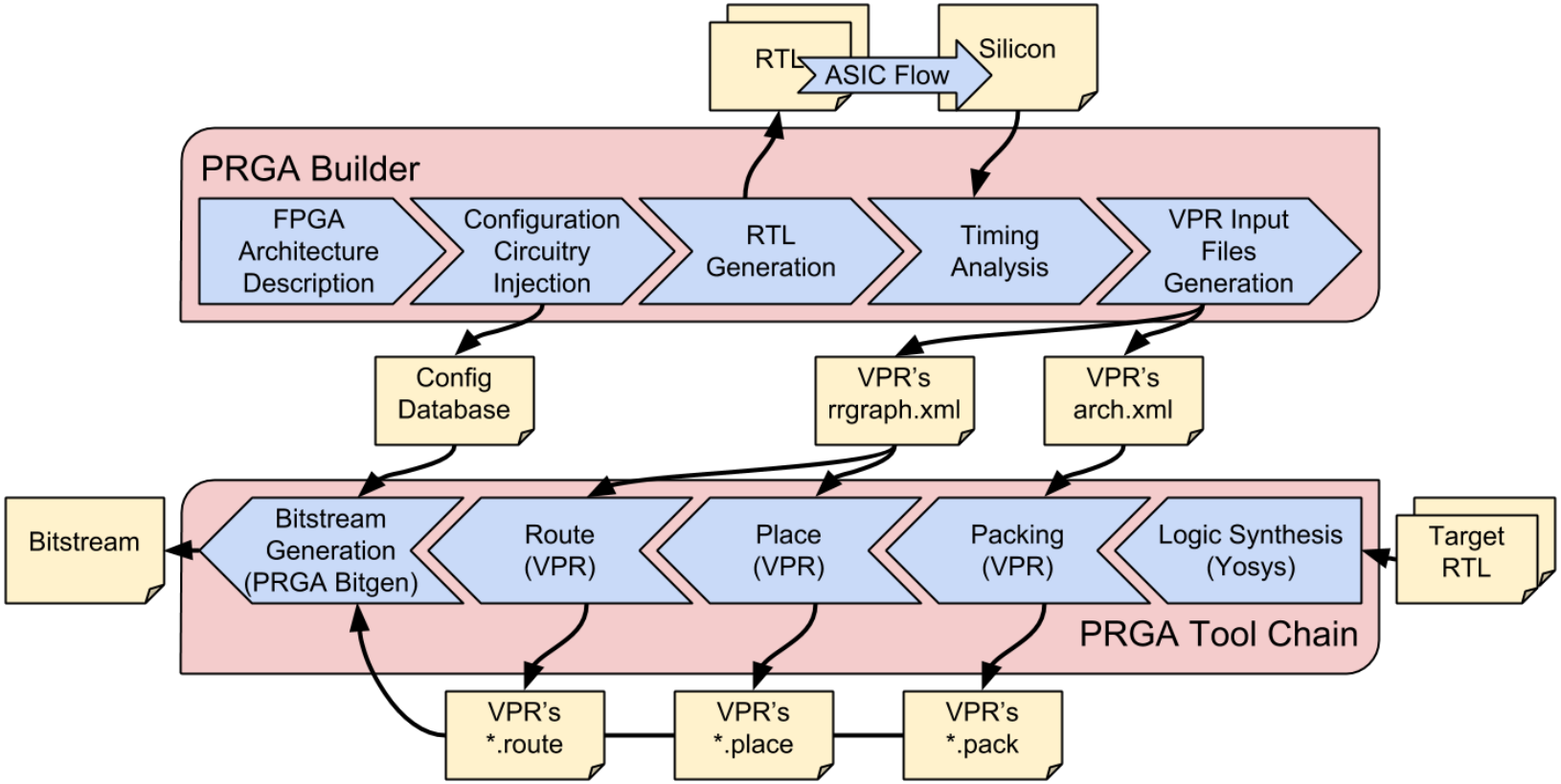
Figures from:

- S. Wilton. (1997). Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memories. *PhD thesis, University of Toronto*
- Compton, K., & Hauck, S. (2002). Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys*, *34*(2), 171–210

Outline

- PRGA Overview
- PRGA Builder: *Python API for building FPGAs*
- PRGA Tool Chain: *open-source tools and generated scripts for using FPGAs built with PRGA Builder*
- Demo
- Evaluation
- Alpha Release

PRGA Workflow



PRGA Goals

- Capable of building commercial-class FPGAs and more
  - **Customizable**: heterogeneous blocks with different sizes, hard IP cores, custom routing resources & connectivity, etc.
  - **Scalable**: hierarchical design with millions of logic elements
- Flexible and extensible enough to support …
  - Custom configuration circuitry: SRAM, multi-context, time-multiplexed, etc.
  - Custom RTL generation and (semi-) custom ASIC flow
  - Other additional/replaceable steps in the flow

PRGA Builder

Python API for RTL & script generation

Examples available:

examples/fpga/*/build.py

```
# Create an ArchitectureContext
#   ArchitectureContext is the entrance to all architecture description
#   APIs, and the container of all data of a custom FPGA
ctx = ArchitectureContext()

# Create some wire segments
ctx.create_segment(name = 'L1', width = 12, length = 1)

# Create a global wire
clk = ctx.create_global(name = 'clk', is_clock = True)

# Create one CLB type
clb = ctx.create_logic_block(name = 'CLB')
# Add ports to this CLB
clb.add_input (name = 'I',   width = 8, side = Side.left)
clb.add_output(name = 'O',   width = 2, side = Side.right)
clb.add_clock (name = 'CLK',            side = Side.bottom,
               global_ = 'clk')
for i in range(2):
    # Add logic elements (primitives) to this CLB
    clb.add_instance(name = 'LUT'+str(i),
                     model = ctx.primitives['lut4'])
    clb.add_instance(name = 'FF'+str(i),
                     model = ctx.primitives['flipflop'])
    # Add configurable intra-block connections to this CLB
    clb.add_connections(
            sources = clb.instances['LUT'+str(i)].pins['out'],
            sinks = clb.instances['FF'+str(i)].pins['D'],
            pack_pattern = True)
    clb.add_connections(
            sources = clb.instances['LUT'+str(i)].pins['out'],
            sinks = clb.ports['O'][i])
    clb.add_connections(
            sources = clb.ports['CLK'],
            sinks = clb.instances['FF'+str(i)].pins['clk'])
    clb.add_connections(
            sources = clb.instances['FF'+str(i)].pins['Q'],
            sinks = clb.ports['O'][i])

clb.add_connections(
        sources = clb.ports['I'],
        sinks = [clb.instances['LUT0'].pins['in'],
                 clb.instances['LUT1'].pins['in']])
```

PRINCETON UNIVERSITY

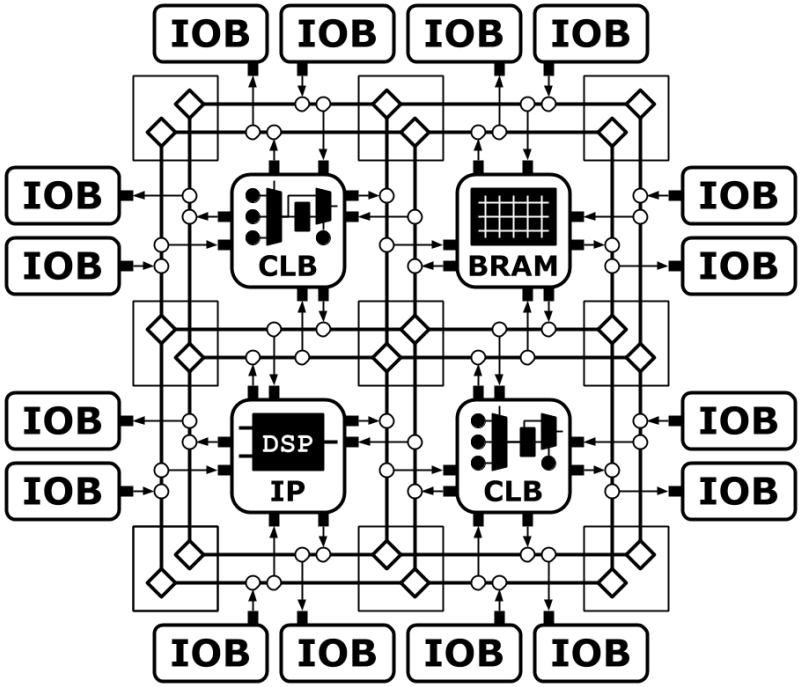
github.com/PrincetonUniversity/prga

PRGA Builder: Architecture Customizability

Blocks:

- ✓Heterogeneous blocks
- ✓Blocks with different sizes
- ✓Custom IP cores

- ❑Multi-mode primitives
  - ❑Fracturable LUT
  - ❑Fracturable MAC
  - ❑LUT/Distributed BRAM

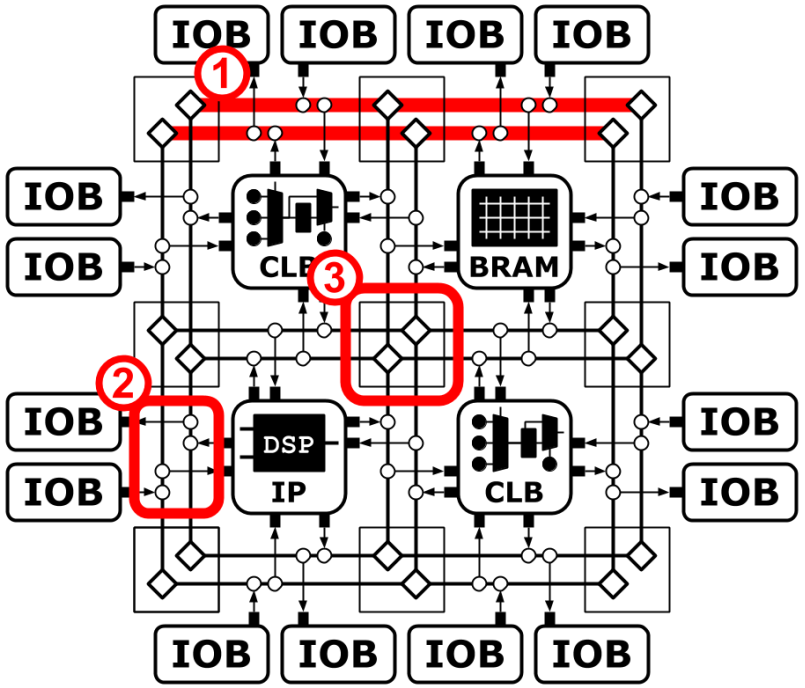


PRGA Builder: Architecture Customizability

Wire Resources (①):

- ✓Wire segments with different lengths
- ✓Global wires

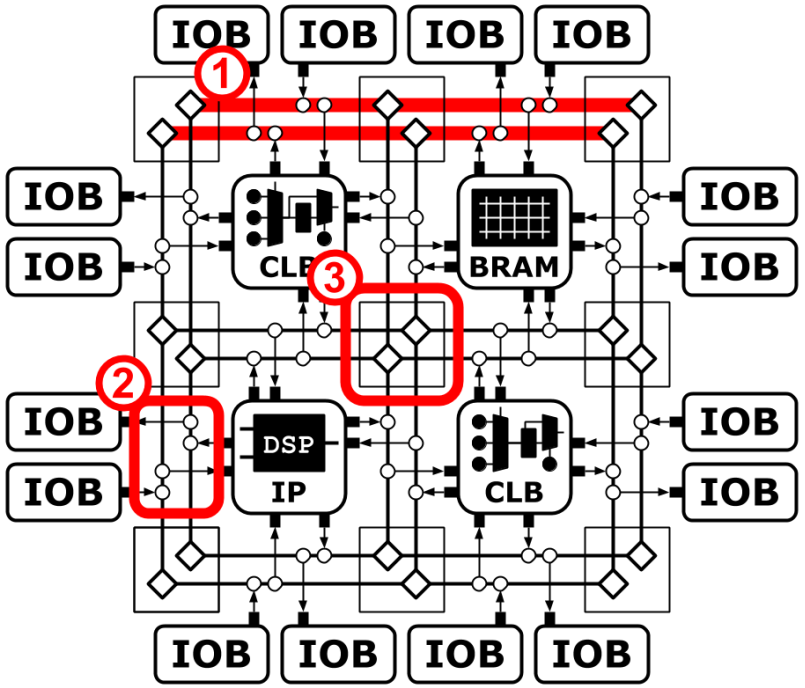
- ❑Irregular channels
- ❑Curved wires



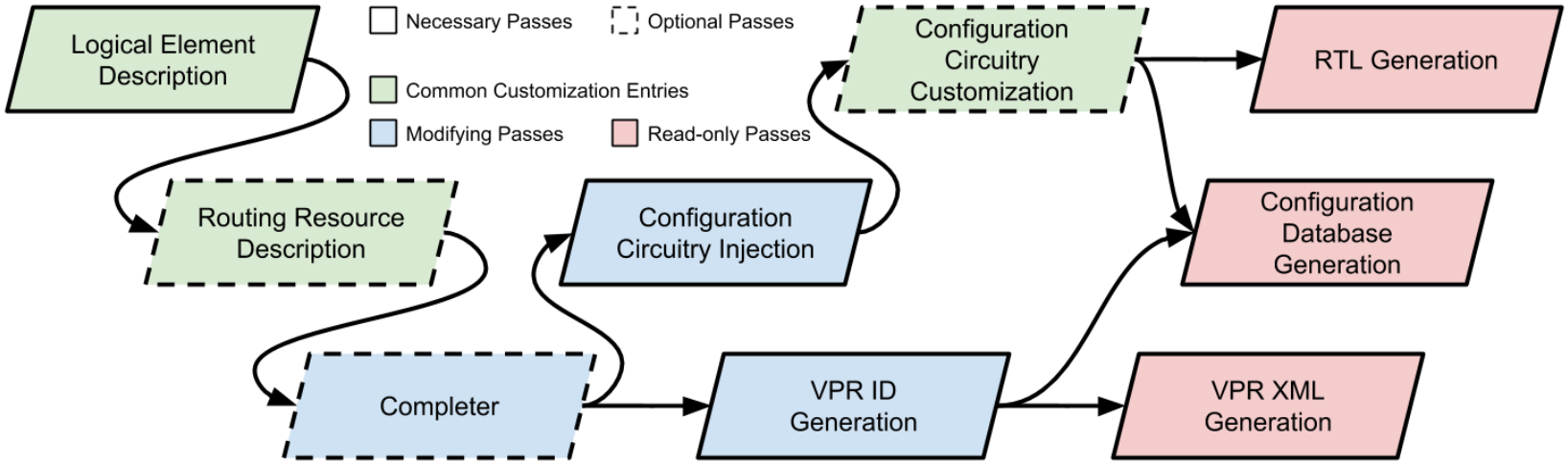
PRGA Builder: Architecture Customizability

Connection Blocks (②) and Switch Blocks (③):

- ✓Fully customizable connectivity
- ✓Different routing blocks at each location



PRGA Builder: Pass-based Generation Flow



- Generation flow based on **modularized**, **replaceable**, **extensible** passes
- Centralized "architecture context"

PRGA Builder: Configuration Circuitry & Database

- More than just configuration memory
  - Flipflop chain-based
  - Supporting DPGA/Tabular-style and more
  - [WIP] Latch array-based
- Protobuf[1]-based, extensible database
  - More than bit offsets and values
  - Performance & readability
- [Future] Compatible with Symbiflow FASM[2]

[1] Google, “Protocol Buffers.” https://developers.google.com/protocol-buffers/
[2] SymbiFlow, “FASM” https://github.com/SymbiFlow/fasm, 2018

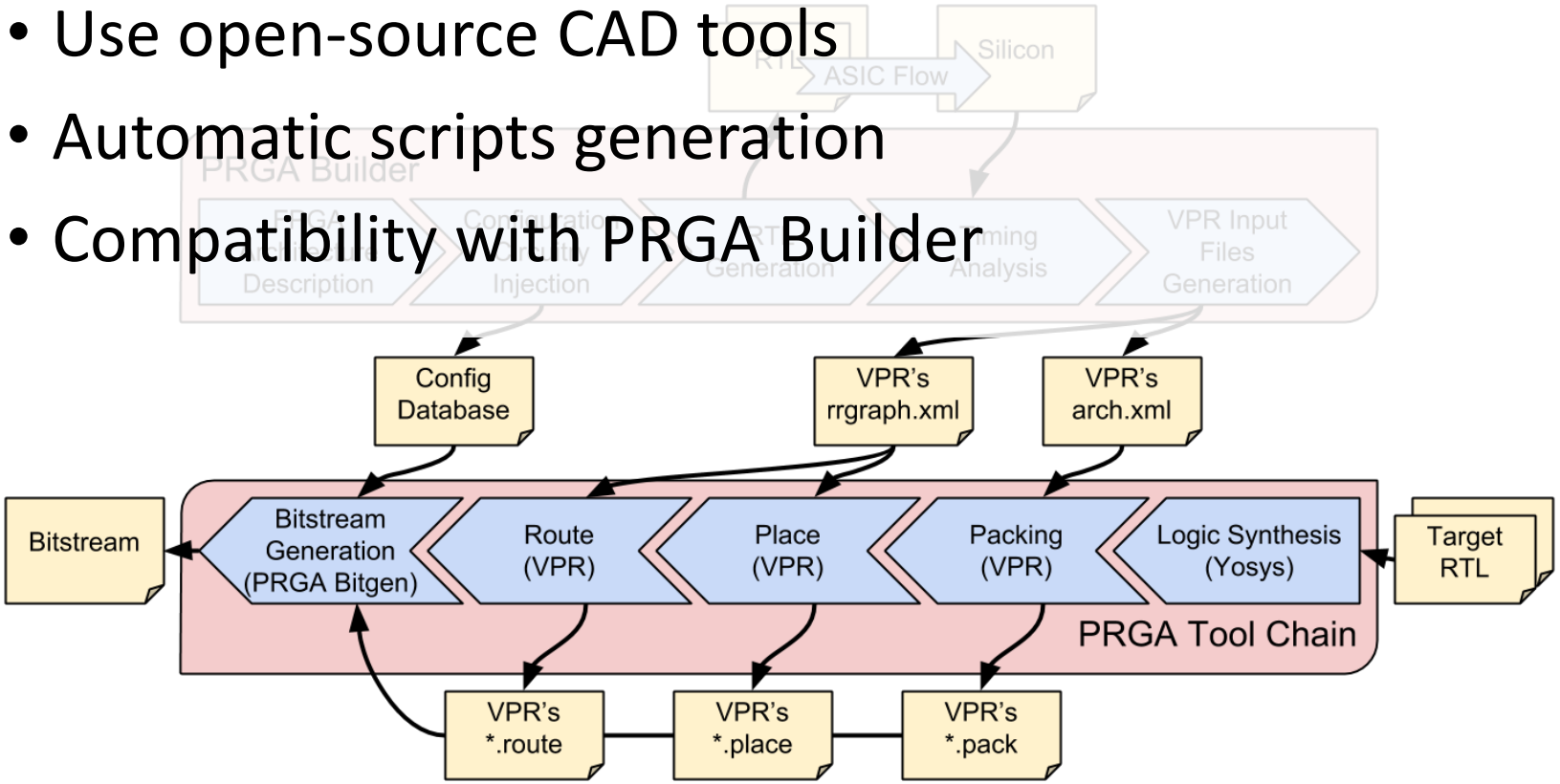
PRGA Builder: RTL Generation & Physical Layout

- Jinja2[1]-based templated RTL generation
  - Replace templates to customize RTL generation
- [Release v0.2] Hierarchical module organization
- [WIP] Example scripts for ASIC flow
  - Semi-custom ASIC flow supported
  - Automatic timing extraction from STA tools

[1] A. Ronacher, “Jinja2.” http://jinja.pocoo.org/, 2014

PRGA Tool Chain

- Use open-source CAD tools
- Automatic scripts generation
- Compatibility with PRGA Builder



PRGA Tool Chain: Synthesis & Place'n'Route

- Synthesis: **Yosys**[1]
  - [WIP] Yosys script generation
- Place'n'Route: **VPR**[2]
  - Architecture description & routing resource graph generation
  - Following latest commits on Github, ready for release 8

[1] C. Wolf, "Yosys Open SYnthesis Suite." http://www.clifford.at/yosys/
[2] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," ACM Trans. Reconfigurable Technol. Syst., vol. 7, pp. 6:1–6:30, July 2014

PRGA Tool Chain: Bitstream Generation

- **PRGA Bitgen:** a C++ framework for bitstream generators compatible with PRGA Builder
  - Extensible configuration database parser
  - Generates bitstream based on VPR outputs
  - [Future] Compatible with SymbiFlow FASM

PRGA Tool Chain: Additional Tools

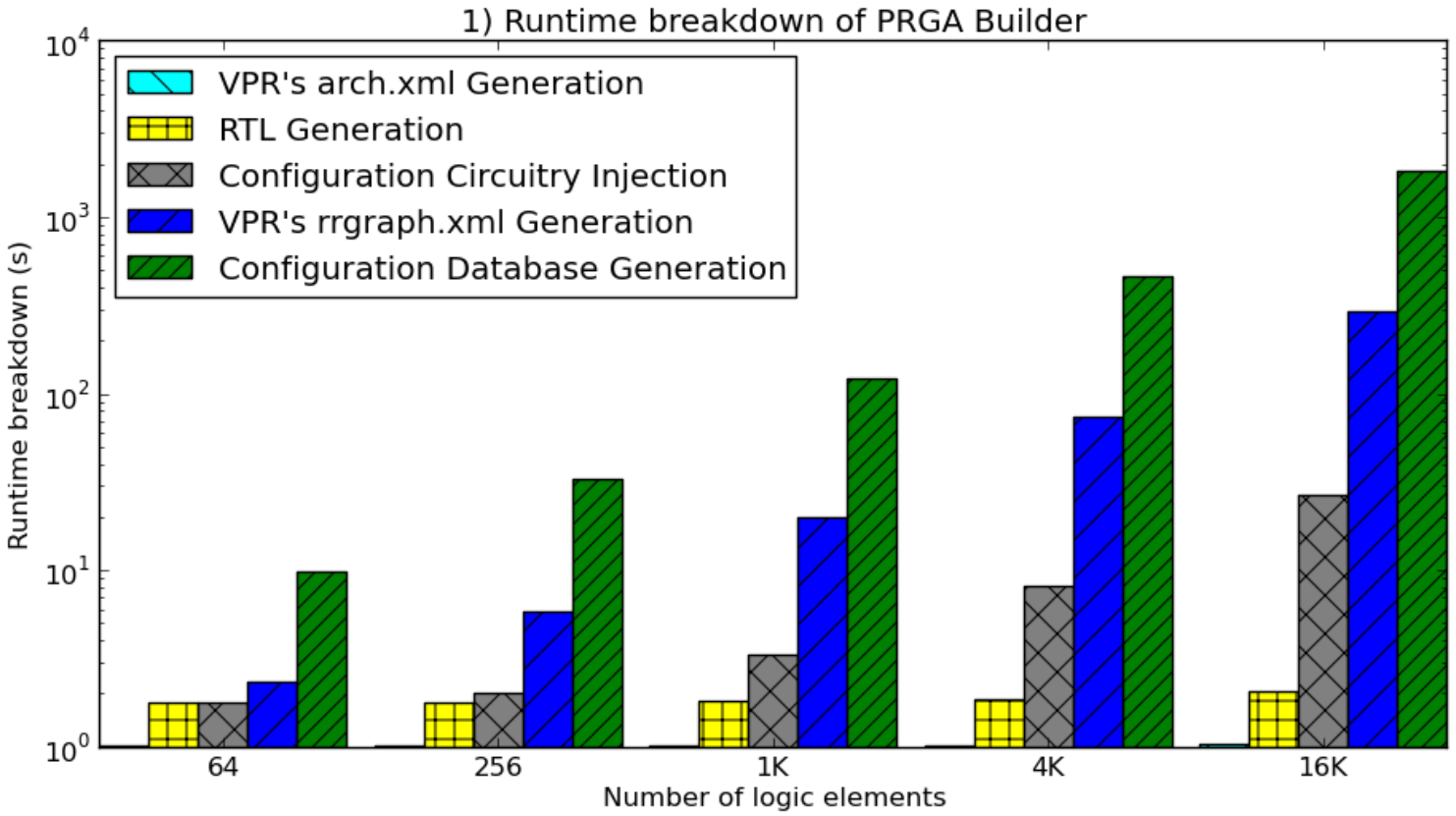
- **Simproj**: simulation project generator
  - Generates testbench, Makefile, yosys scripts
  - Complete Verilog-to-bitstream flow
  - Simulates programming process or loads bitstream with Verilog hacks

Examples available: examples/fpga/*/build.py

PRGA Demo

- Architecture settings
  - 6x6 CLB, each with 2 LUT4 and 2 DFF
  - 24 unit-length wire segments per channel
  - Organized as 3x3 macro-tiles
- Target design: BCD2BIN converter
  - 3-state FSM
  - Handshake interface

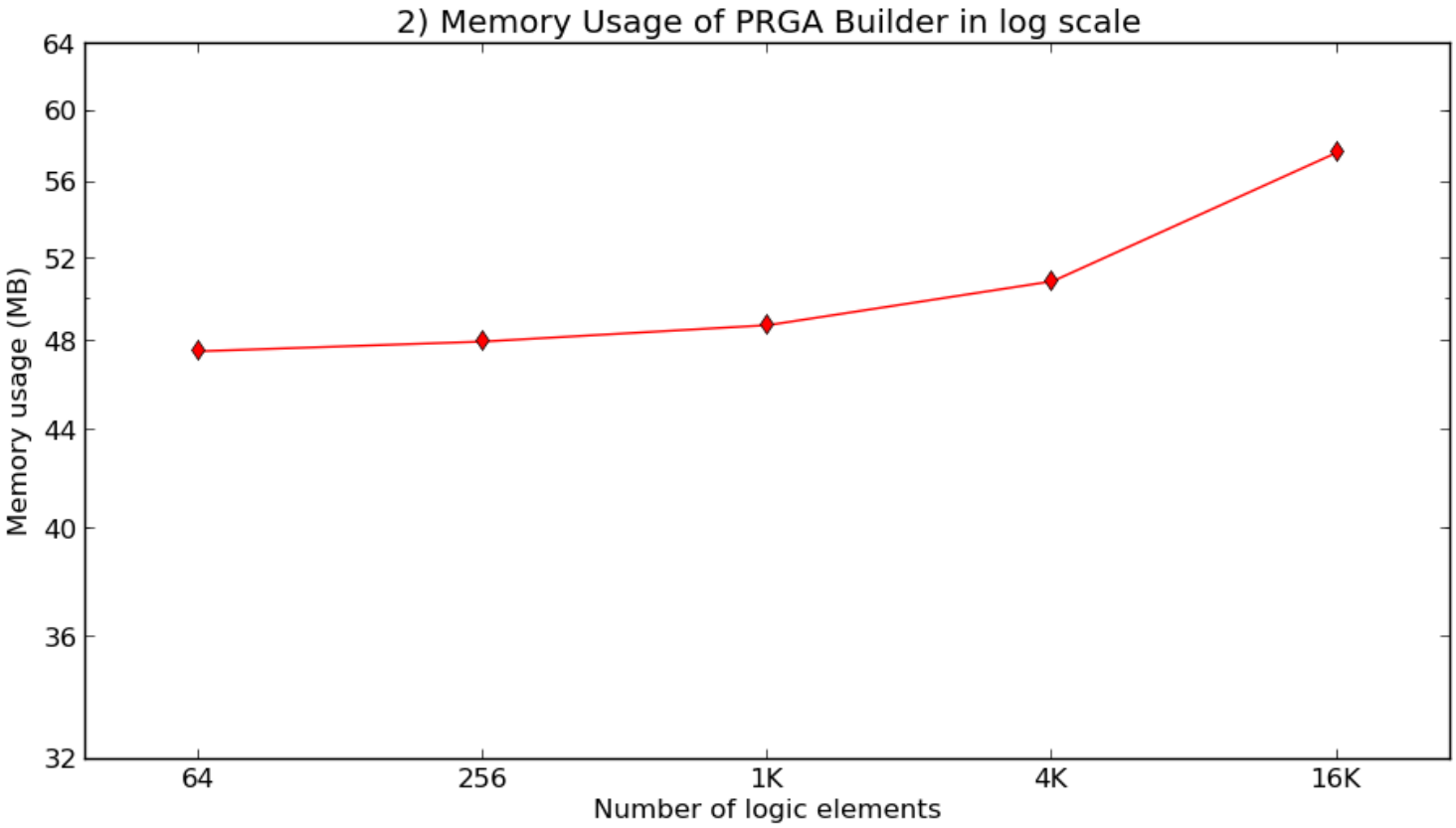
Evaluation:
Runtime Breakdown of PRGA Builder



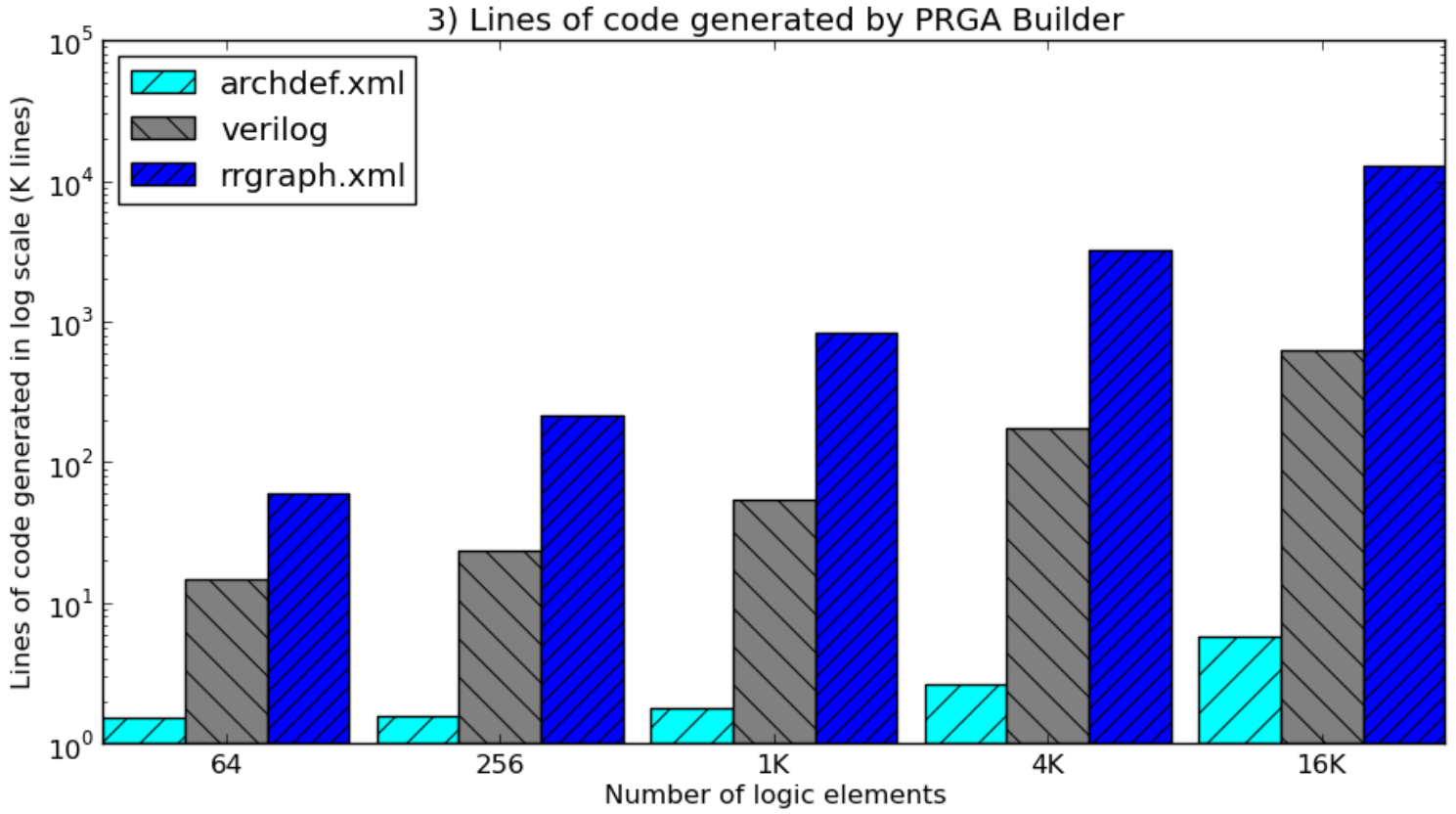
PRINCETON UNIVERSITY

github.com/PrincetonUniversity/prga

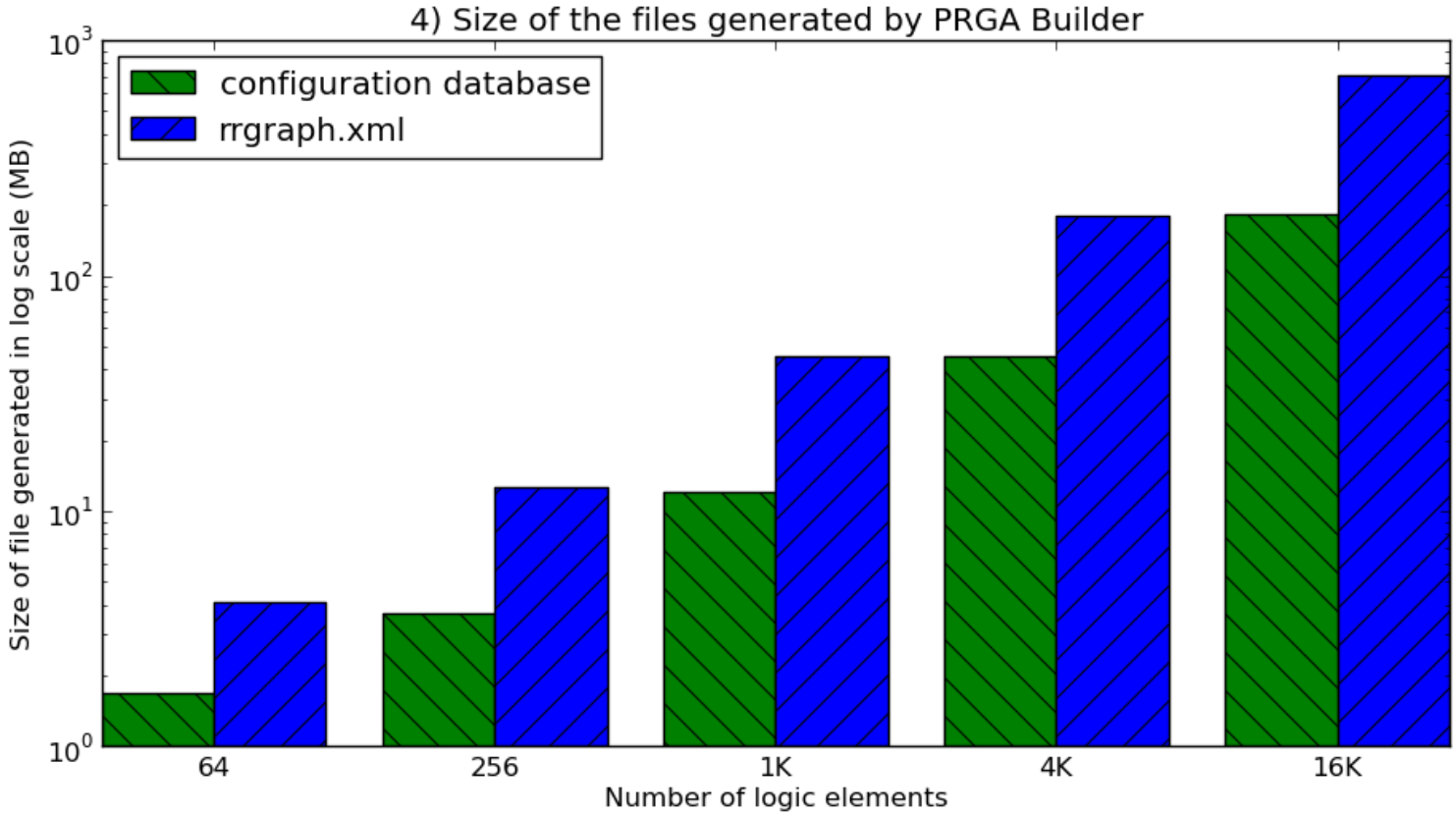
Evaluation: Memory Usage of PRGA Builder



Evaluation:
Lines of Code Generated



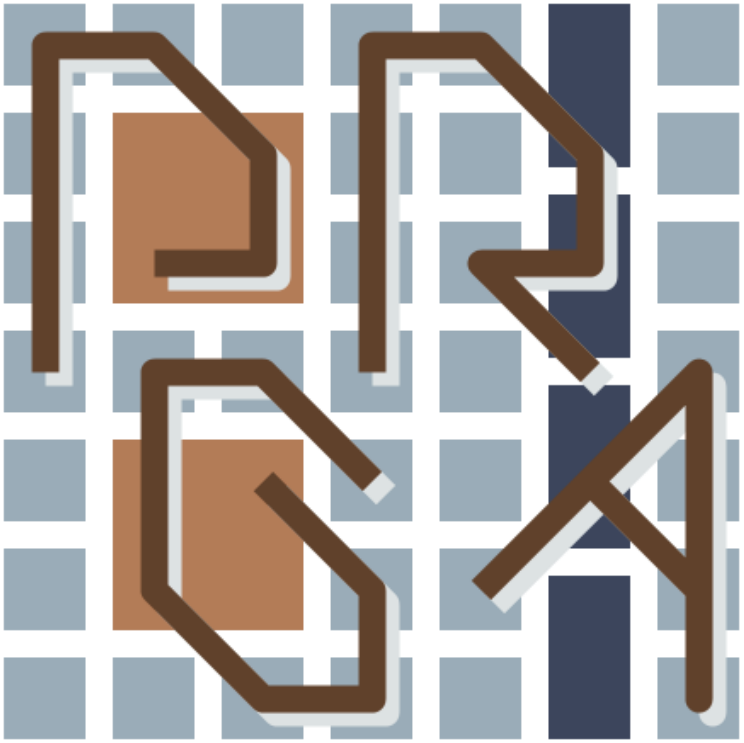
Evaluation:
Size of Generated Files



PRINCETON UNIVERSITY

github.com/PrincetonUniversity/prga

PRGA Alpha Release



Source code & examples:

github.com/PrincetonUniversity/prga

Documentation:

prga.readthedocs.io

Princeton Reconfigurable Gate Array