PRGA: An Open-source Framework for Building and Using Custom FPGAs

Ang Li, David Wentzlaff
Princeton University

github.com/PrincetonUniversity/prga
The Era of Open-Source Hardware

A New Golden Age for Computer Architecture
Agriculture Technology
Monitoring Noise Pollution
The Computational Sprinting Game
Blockchain from a Distributed Computing Perspective

NVDLA
MIAOW
OpenPiton
RISC-V
OpenSPARC
OpenRISC
PULP Ariane

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.)

github.com/PrincetonUniversity/prga
Enabled Applications

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

Figures from:
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

[github.com/PrincetonUniversity/prga]
PRGA Workflow

PRGA Builder
- FPGA Architecture Description
- Configuration Circuitry Injection
- RTL Generation
- Timing Analysis
- VPR Input Files Generation

PRGA Tool Chain
- Bitstream Generation (PRGA Bitgen)
- Route (VPR)
- Place (VPR)
- Packing (VPR)
- Logic Synthesis (Yosys)

Silicon

RTL

ASIC Flow

github.com/PrincetonUniversity/prga
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
# 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']])

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

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

[Diagram of PRGA Builder with IOB, CLB, BRAM, DSP, and IP blocks]
PRGA Builder: Architecture Customizability

**Wire Resources (1):**
- ✔ Wire segments with different lengths
- ✔ Global wires
- ❑ Irregular channels
- ❑ Curved wires

[Image of PRGA Builder diagram]

[GitHub link: github.com/PrincetonUniversity/prga]
PRGA Builder: Architecture Customizability

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

✓ Fully customizable connectivity
✓ Different routing blocks at each location

github.com/PrincetonUniversity/prga
PRGA Builder: Pass-based Generation Flow

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

[Diagram of PRGA Builder with flow from Logical Element Description to RTL Generation through Routing Resource Description, Completer, VPR ID Generation, VPR XML Generation, and Configuration Circuitry Customization.]

github.com/PrincetonUniversity/prga
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]

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

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

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

github.com/PrincetonUniversity/prga
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](https://github.com/PrincetonUniversity/prga)
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

github.com/PrincetonUniversity/prga
Evaluation:
Runtime Breakdown of PRGA Builder

1) Runtime breakdown of PRGA Builder

- VPR's arch.xml Generation
- RTL Generation
- Configuration Circuitry Injection
- VPR's rrgraph.xml Generation
- Configuration Database Generation

Runtime breakdown (s)

Number of logic elements

64 256 1K 4K 16K

github.com/PrincetonUniversity/prga
Evaluation:
Memory Usage of PRGA Builder

2) Memory Usage of PRGA Builder in log scale

Number of logic elements

Memory usage (MB)

64 256 1K 4K 16K

32 36 40 44 48 52 56 60 64
Evaluation: 
Lines of Code Generated

3) Lines of code generated by PRGA Builder

![Bar chart showing lines of code generated by PRGA Builder for different number of logic elements (64, 256, 1K, 4K, 16K). The chart uses a log scale for the y-axis, ranging from $10^0$ to $10^5$. The chart compares three formats: archdef.xml, verilog, and rgraph.xml.](image_url)
Evaluation: Size of Generated Files

4) Size of the files generated by PRGA Builder

<table>
<thead>
<tr>
<th>Number of logic elements</th>
<th>configuration database</th>
<th>rgraph.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>256</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1K</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>4K</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>16K</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
PRGA Alpha Release

Source code & examples: github.com/PrincetonUniversity/prga

Documentation: prga.readthedocs.io
Princeton Reconfigurable Gate Array