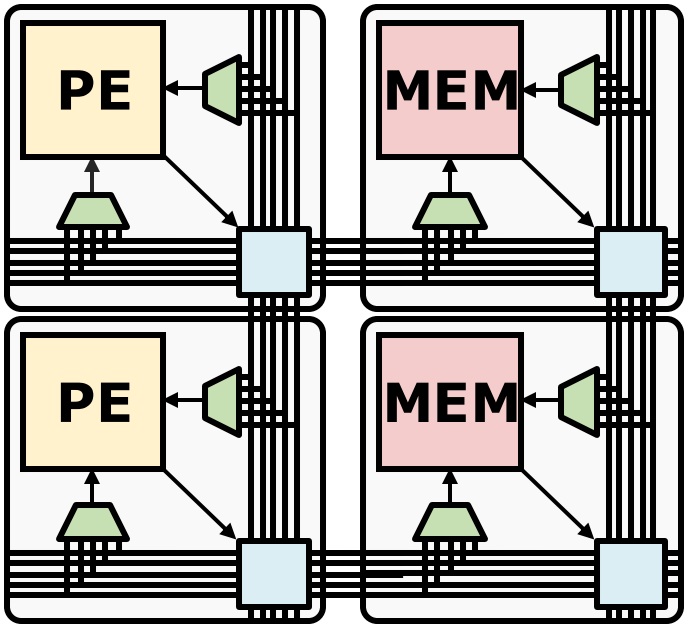
Canal: A Flexible Interconnect Generator for Coarse-Grained Reconfigurable Arrays

Keyi Zhang & Jackson Melchert

Stanford University

Motivation - Interconnect Specification

- We have DSLs for specifying the PEs and memory tiles
  - How do we specify the configurable interconnect that makes up the rest of the CGRA?
- Canal is an interconnect specification DSL that greatly simplifies the creation of CGRAs
  - Enables easy design space exploration of CGRA interconnect architecture



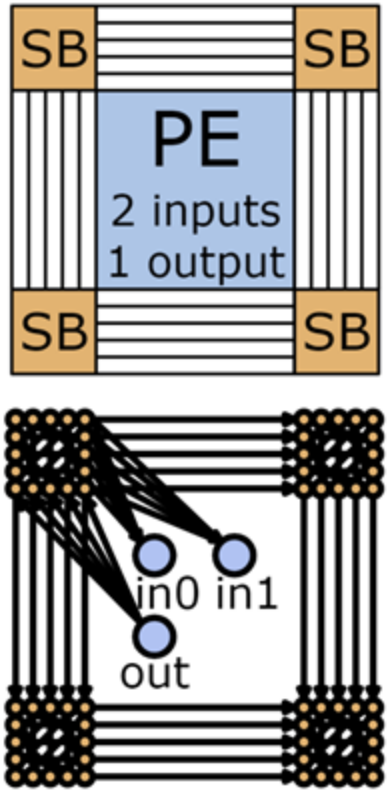
Contributions

- Canal is a Python-embedded domain-specific language and a CGRA oriented interconnect generator:
  - A graph-based intermediate representation (IR) for CGRA interconnects
  - A compiler that generates hardware from the IR
  - A configuration management tool that takes the IR and produces a bitstream
  - A design-space exploration tool

Specifying a Configurable Interconnect

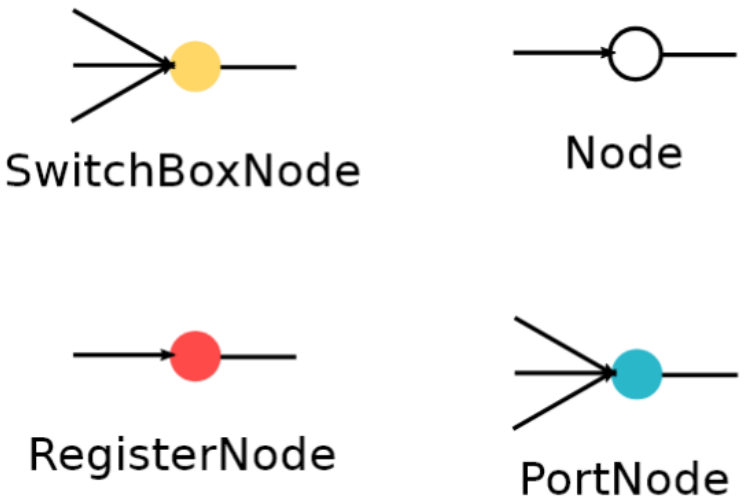
- Specify the interconnect as a graph
  - Nodes are ports of tiles, branching points, or multiplexers
  - Directed edges are wires

- From this graph we can automatically generate:
  1. Hardware
  2. Place and route collateral
  3. A bitstream generator



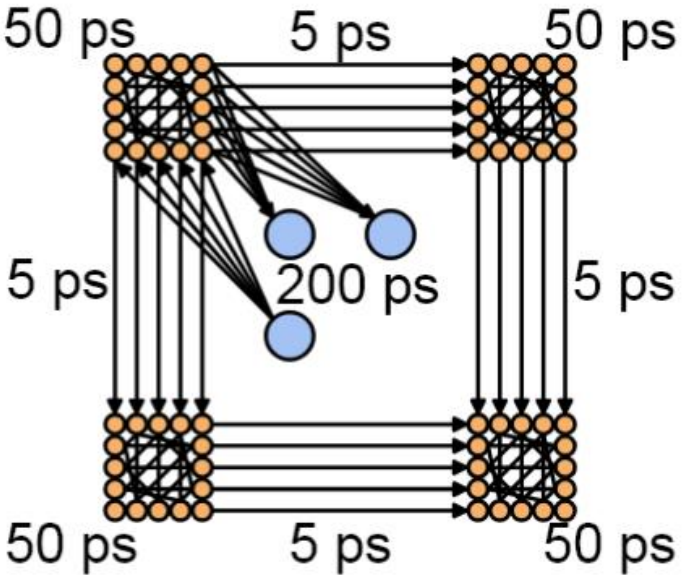
Specifying a Configurable Interconnect

- Each node can have different types, attributes, and other metadata
- Edges can also contain metadata, such as wire delay, which is used for PnR
- Edges are ordered



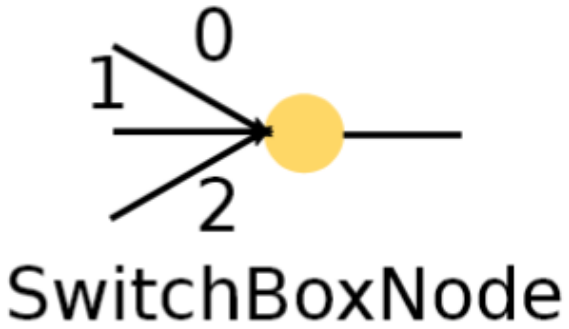
Specifying a Configurable Interconnect

- Each node can have different types, attributes, and other metadata
- Edges can also contain metadata, such as wire delay, which is used for PnR
- Edges are ordered



Specifying a Configurable Interconnect

- Each node can have different types, attributes, and other metadata
- Edges can also contain metadata, such as wire delay, which is used for PnR
- Edges are ordered



Canal Language

- The Canal language is a Python-embedded domain-specific language that constructs the interconnect intermediate representation:

Node and edge creation:

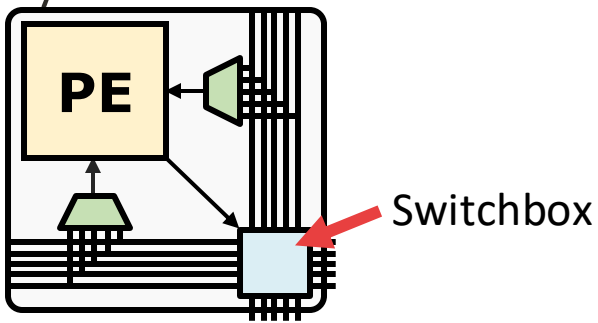
```
node = Node(x=1, y=1, track=1)
for port_node in tile.pe.inputs():
    node.add_edge(port_node)
```

High level interconnect generation:

```
create_uniform_interconnect(width=32, height=32, sb_type="wilton",
                            num_tracks=5, track_width=16, reg_density=1)
```

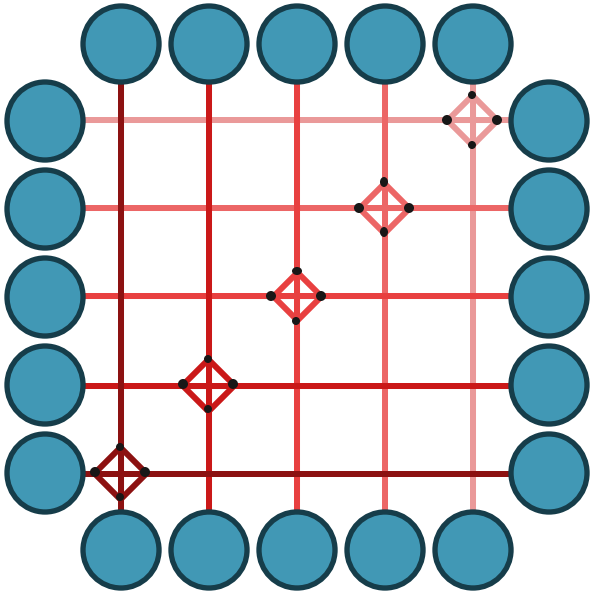
Demo

- In this demo, we'll use Canal to design a switchbox
- Switchboxes are an important part of the reconfigurable interconnect of a CGRA or FPGA
  - They route data from one location on the array to another location
- Switchboxes can have a variety of different topologies
  - Different incoming tracks to the switchbox can route data to the other tracks lots of different ways



Demo

- The topology we will create today is called a disjoint topology
- Each incoming connection can route to each of the three outgoing connections



Demo

- /aha/canal_demo.py
- First, we create the port nodes for incoming data:

```
from canal.cyclone import *
num_tracks = 5

input_nodes = {}
for side in SwitchBoxSide:
    input_nodes[side] = []
    for track in range(num_tracks):
        input_nodes[side].append(
            SwitchBoxNode(0, 0, track, 16, side, SwitchBoxIO.SB_IN))
```

Demo

- Similarly, we also create the output ports from the switchbox

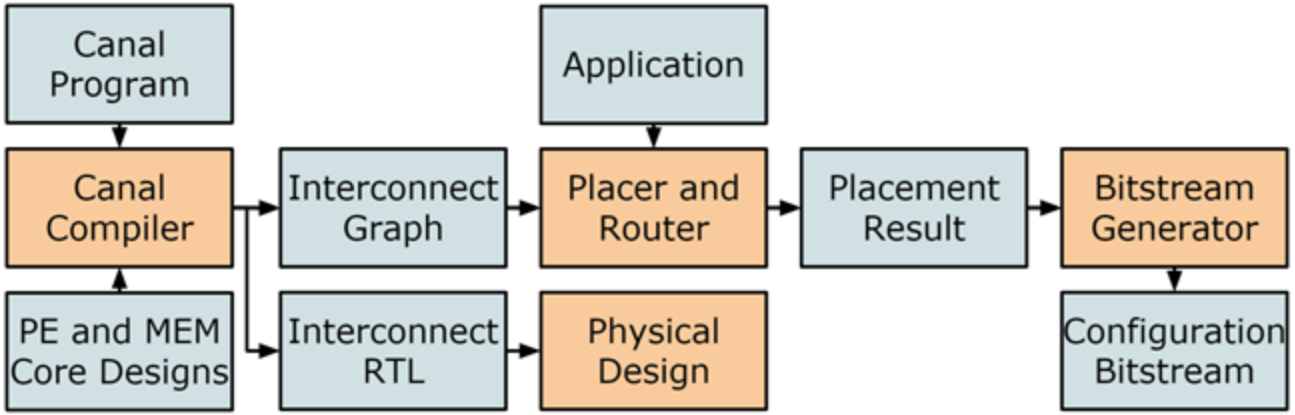
```
output_nodes = {}
for side in SwitchBoxSide:
    output_nodes[side] = []
    for track in range(num_tracks):
        output_nodes[side].append(
          SwitchBoxNode(0, 0, track, 16, side, SwitchBoxIO.SB_OUT))
```

Demo

- Finally, we connect each of the input nodes to output nodes of the same track in each of the three remaining directions

```
for track in range(num_tracks):
    for side_from in SwitchBoxSide:
        for side_to in SwitchBoxSide:
            if side_from == side_to:
                continue
            input_node = input_nodes[side_from][track]
            output_node = output_nodes[side_to][track]
            print(f"Wire {input_node} -> {output_node}")
            input_node.add_edge(output_node)
```

Canal Interconnect Generation System



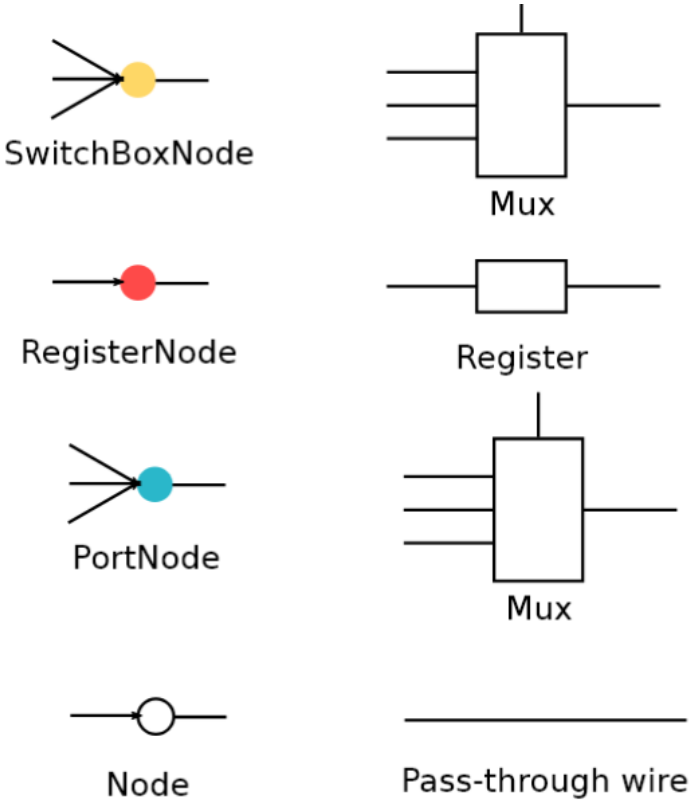
- Fully automated hardware, PnR, and bitstream configuration generation

Generating Interconnect Hardware

1. Nodes with hardware attributes (PEak PEs or Lake memories) instantiate the specified hardware

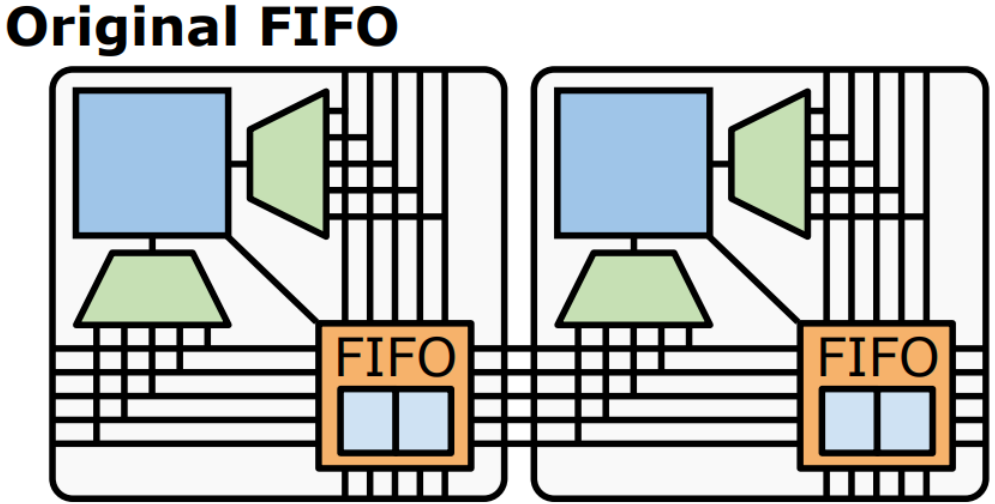
2. Directed edges are translated into wires

3. Nodes with multiple incoming edges generate multiplexers



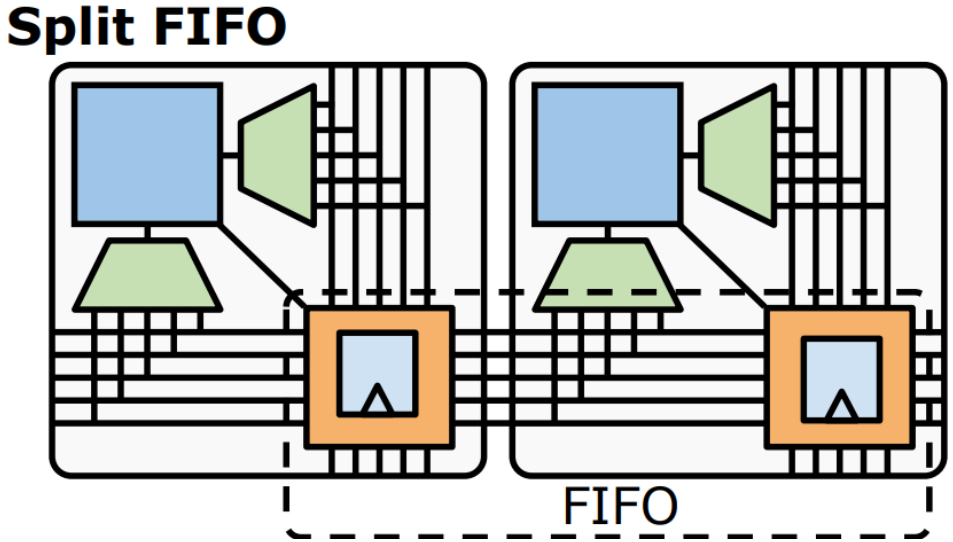
Split FIFO Optimization

- While it is easy to generate a fixed-size FIFO for buffering data throughout the interconnect, the cost is very high
- Using Canal, we can investigate more efficient ways of implementing these buffers

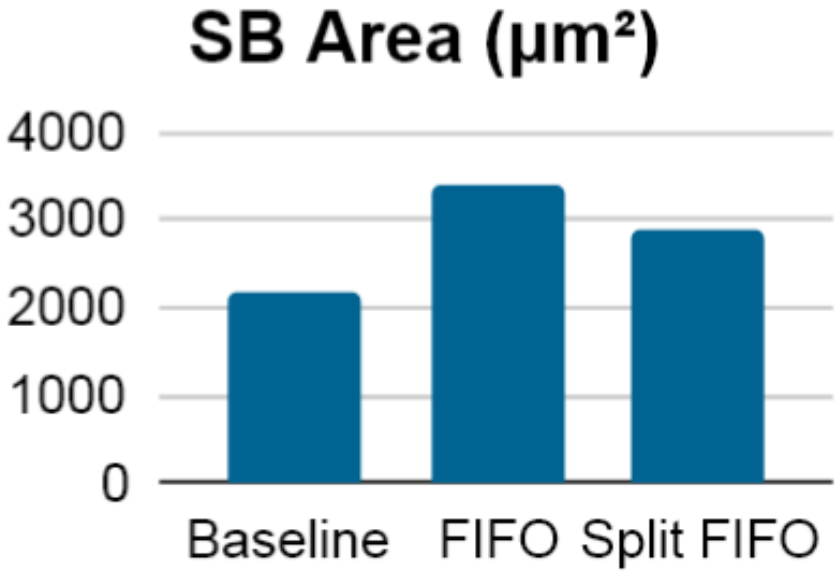


Split FIFO Optimization

- We split the size-two FIFO into pairs of registers in adjacent tiles
- The first register's control signals are passed from the first tile to the second tile

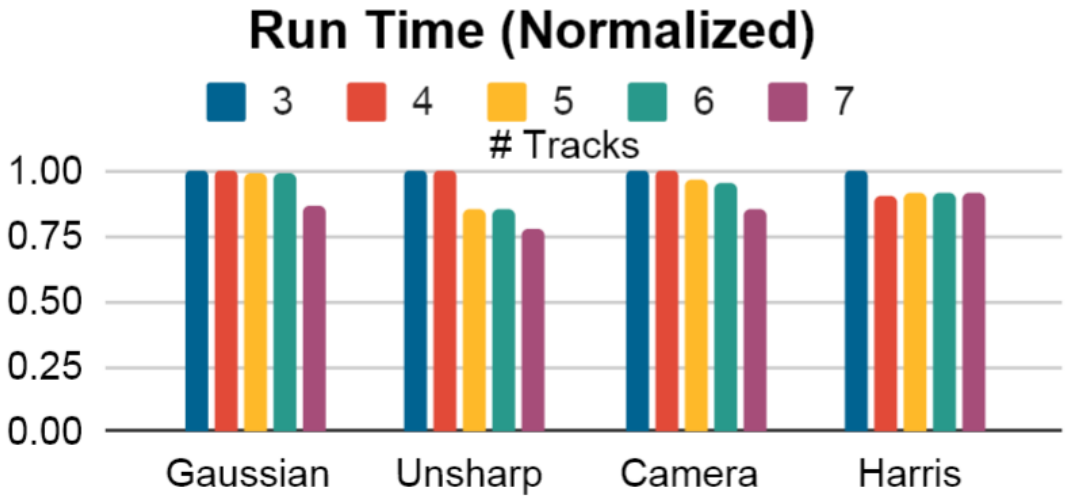


Split FIFO Optimization Results



- Split FIFOs reduce the area cost of buffering data significantly

Design Space Exploration with Canal



- Application run times decrease with more routing resources

Conclusion

- The Canal DSL and IR enable agile design of CGRA interconnects
  - The Canal language is easy to use and intuitive
  - The interconnect generation system automates generating hardware and compiler collateral
  - Design space exploration is easy because of the flexibility afforded by Canal