AutoBridge: Coupling Coarse-Grained Floorplanning with Pipelining for High-Frequency HLS Design on Multi-Die FPGAs

Licheng Guo[1], Yuze Chi[1], Jie Wang[1], Jason Lau[1], Weikang Qiao[1], Ecenur Ustun[2], Zhiru Zhang[2], Jason Cong[1]

University of California Los Angeles[1], Cornell Unversity[2]
lcguo@ucla.edu

https://github.com/Licheng-Guo/AutoBridge

VLSI architecture, synthesis & technology

Problem

- HLS designs often suffer from low frequency
- Hard to fix the problem

```
void kernel(
  float *dram_port0,
  float *result)
{
  ......
}
```

My beautiful C++

```
module kernel()
begin
  wire dram_M_AXI_AVALID
  wire result_S_AXI_AR
  …
end
```

Machine-generated RTL
Hard to read…

```
WARNING: failed to reach
timing target
......

ERROR: routing failed
......
```

???

Reason 1: Abstraction Gap

- HLS has no physical layout information
  - How far will these two registers be apart?
  - How congested will the area be?
- Current HLS relies on inaccurate pre-characterized delay models

Reason 1: Abstraction Gap

- HLS has no physical layout information
  - How far will these two registers be apart?
  - How congested will the area be?
- Current HLS relies on inaccurate pre-characterized delay models

```
void top() {
  temp = foo(...);
  bar(temp, ...);
}
```

Source C++ code

HLS

```
always @ (posedge ap_clk)
  bar_in <= foo_out;
```

HLS registers the connection once
(which looks reasonable)

Reason 1: Abstraction Gap

- HLS has no physical layout information
  - How far will these two registers be apart?
  - How congested will the area be?
- Current HLS relies on inaccurate pre-characterized delay models

```
void top() {
  temp = foo(...);
  bar(temp, ...);
}
```

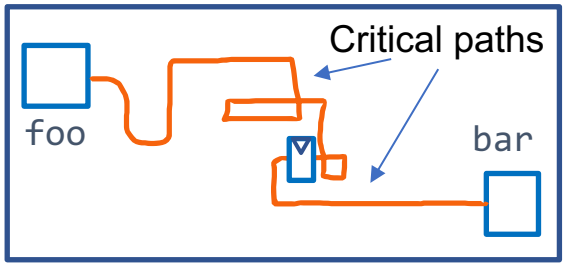
Source C++ code

HLS

```
always @ (posedge ap_clk)
  bar_in <= foo_out;
```

HLS registers the connection once (which looks reasonable)

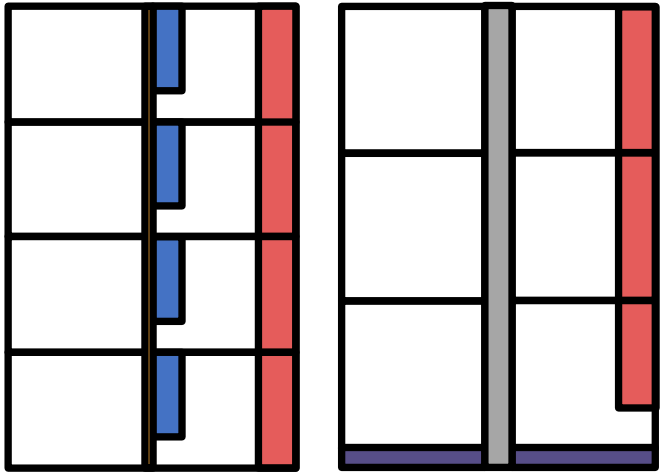
Placer Router



This is possible (and common!)

Reason 2: FPGA Complexity

- FPGAs are increasingly large
- Multiple dies integrated together
- High delay penalty for die-crossing
  - ~ 1ns [Pereira FPGA'14]
- Large IPs with pre-determined location

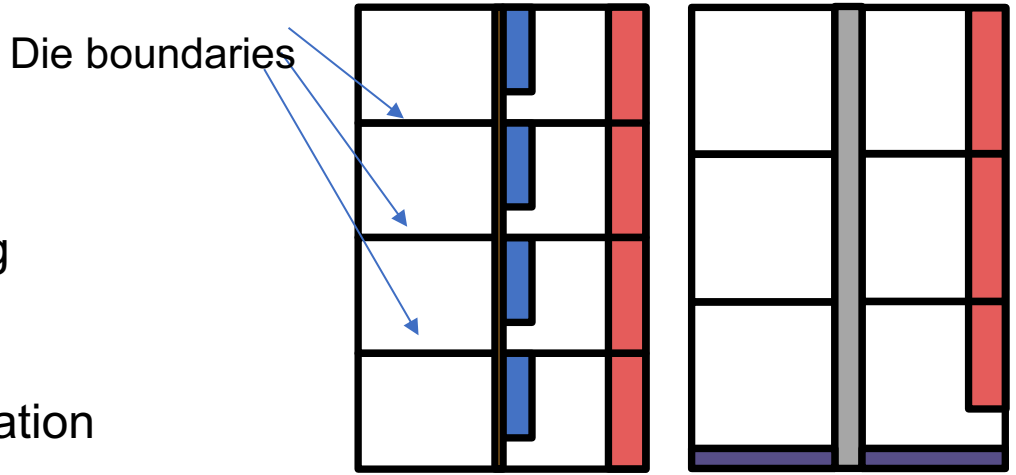


Xilinx Alveo U250

Xilinx Alveo U280

Reason 2: FPGA Complexity

- FPGAs are increasingly large
- Multiple dies integrated together
- High delay penalty for die-crossing
  - ~ 1ns [Pereira FPGA'14]
- Large IPs with pre-determined location

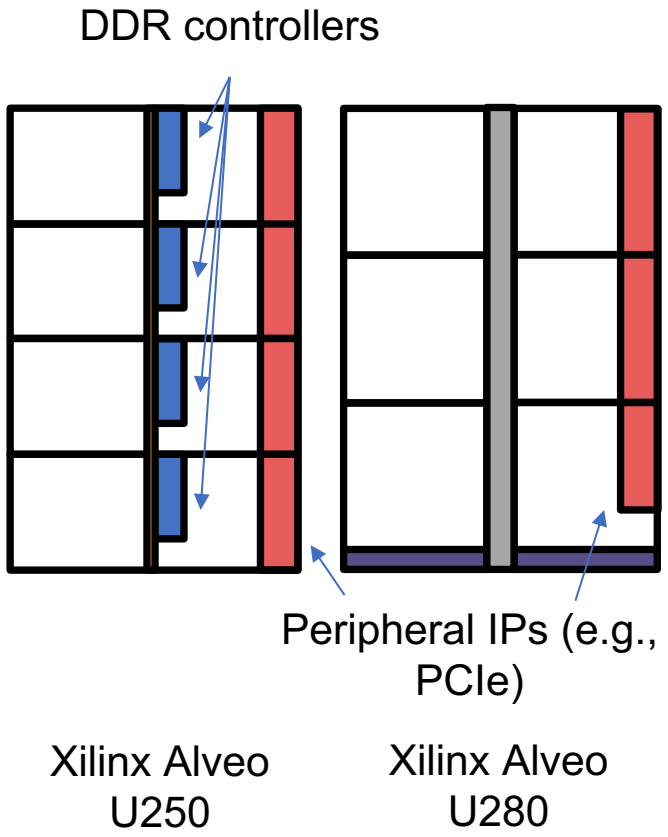


Xilinx Alveo U250

Xilinx Alveo U280

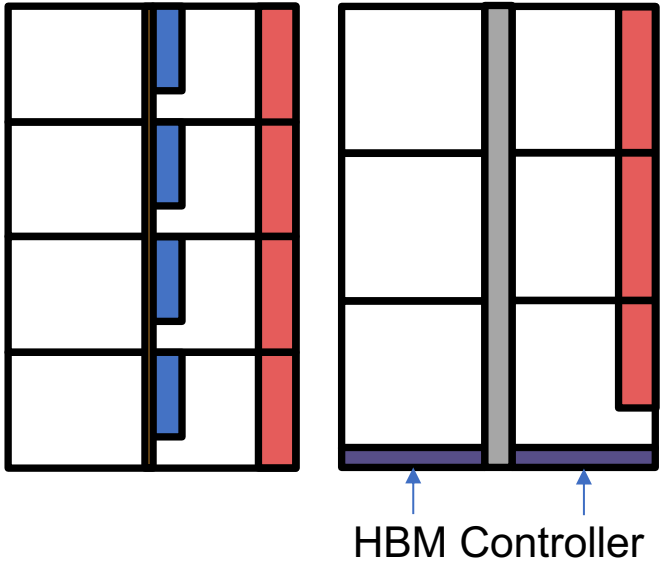
Reason 2: FPGA Complexity

- FPGAs are increasingly large
- Multiple dies integrated together
- High delay penalty for die-crossing
  - ~ 1ns [Pereira FPGA'14]
- Large IPs with pre-determined location



Reason 2: FPGA Complexity

- FPGAs are increasingly large
- Multiple dies integrated together
- High delay penalty for die-crossing
  - ~ 1ns [Pereira FPGA'14]
- Large IPs with pre-determined location

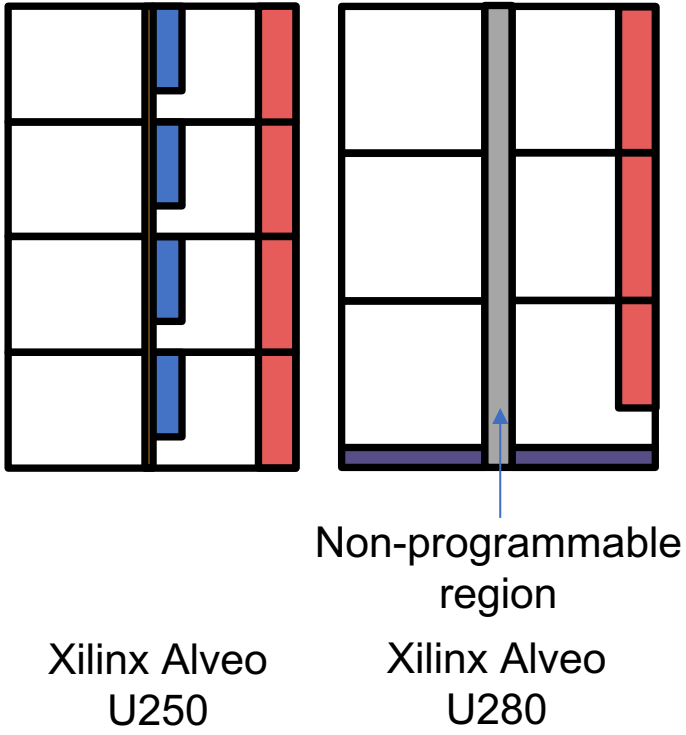


Xilinx Alveo U250

Xilinx Alveo U280

Reason 2: FPGA Complexity

- FPGAs are increasingly large
- Multiple dies integrated together
- High delay penalty for die-crossing
  - ~ 1ns [Pereira-2014]
- Large IPs with pre-determined location

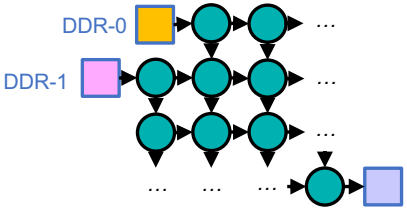


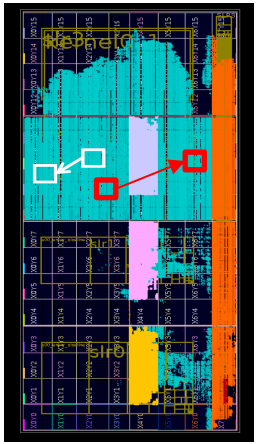
Reason 2: FPGA Complexity

- HLS has limited consideration of those physical barriers

Reason 2:

- HLS has limited consideration of those physical barriers
- Placer often needs to pack things together to reduce die crossing
  - Increase local congestion instead

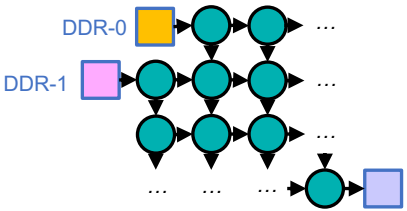


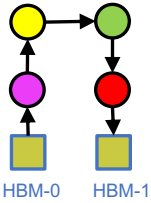


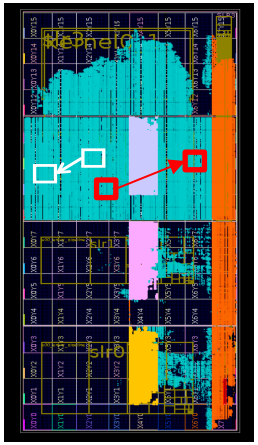
Systolic array on U250

Reason 2:

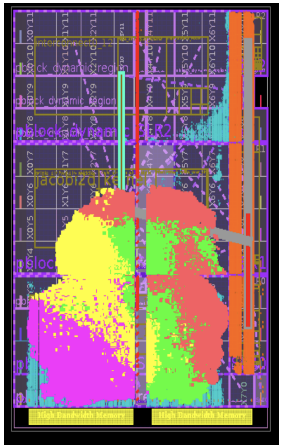
- HLS has limited consideration of those physical barriers
- Placer often needs to pack things together to reduce die crossing
  - Increase local congestion instead
- Sub-optimal choice of crossing wires by the placer / router







Systolic array on U250

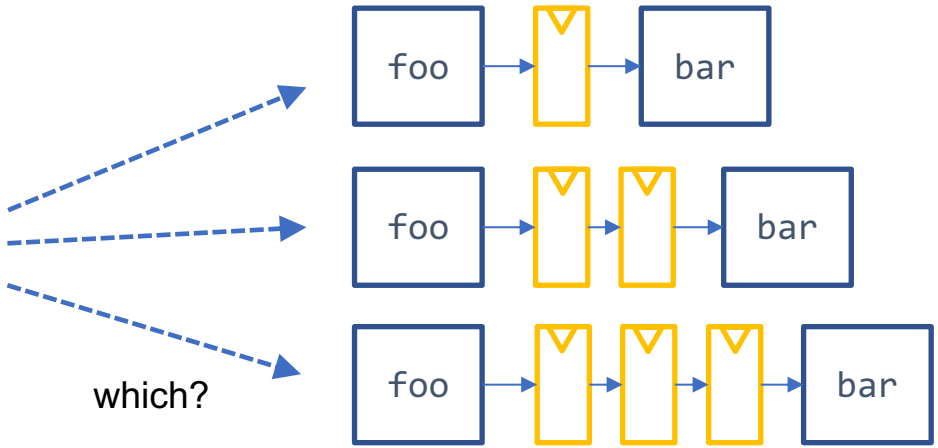


Stencil accelerator on U280

Opportunities and Challenges

- HLS has the freedom to alter the scheduling solution
  - Potentially add more pipelining
- But where and how many?
- Will performance (cycle count) be affected?
- Scalability of the method?

```
void top() {
  temp = foo(...);
  bar(temp, ...);
}
```



Previous Attempts

- Existing efforts focus on fine-grained delay model calibration
  - [Zheng-FPGA'12] Iteratively place & route to calibrate delay information for HLS
  - [Cong-2004] Placement-driven scheduling and binding

Previous Attempts

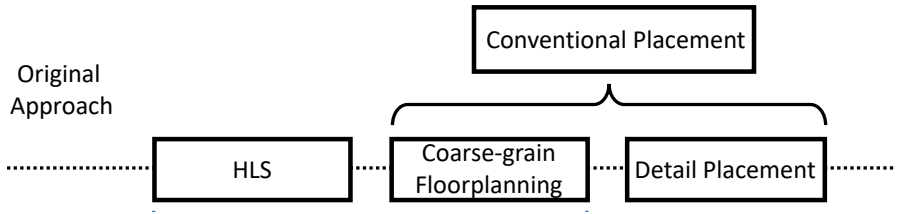
- Existing efforts focus on fine-grained delay model calibration
  - [Zheng-FPGA'12] Iteratively place & route to calibrate delay information for HLS
  - [Cong-2004] Placement-driven scheduling and binding
- Not scalable, limited to tiny designs (only ~1000s of LUTs)
  - Our benchmarks can be 100X larger and many take days to implement

Previous Attempts

- Existing efforts focus on fine-grained delay model calibration
  - [Zheng-FPGA'12] Iteratively place & route to calibrate delay information for HLS
  - [Cong-2004] Placement-driven scheduling and binding
- Not scalable, limited to tiny designs (only ~1000s of LUTs)
  - Our benchmarks can be 100X larger and many take days to implement
- Placer and router may not behave as expected

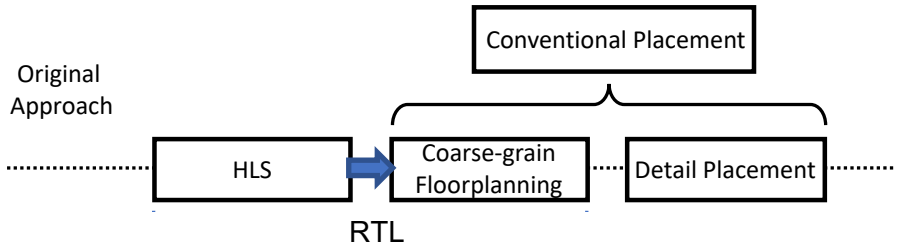
Core Idea

- Floorplan the design during HLS compilation
  - In a coarse granularity
- Add additional pipelining based on floorplan results
  - Guarantee no loss of performance



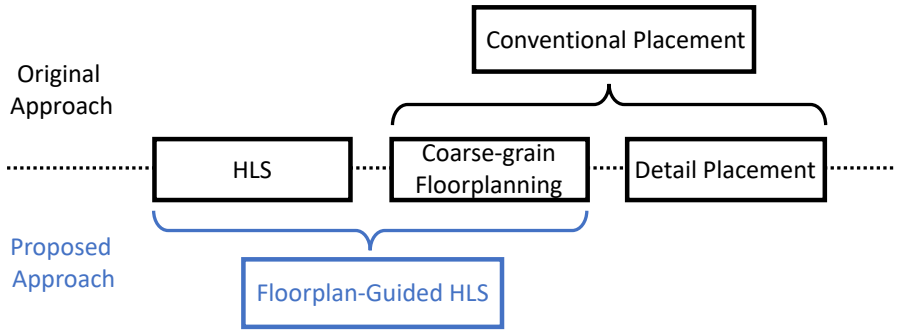
Core Idea

- Floorplan the design during HLS compilation
  - In a coarse granularity
- Add additional pipelining based on floorplan results
  - Guarantee no loss of performance



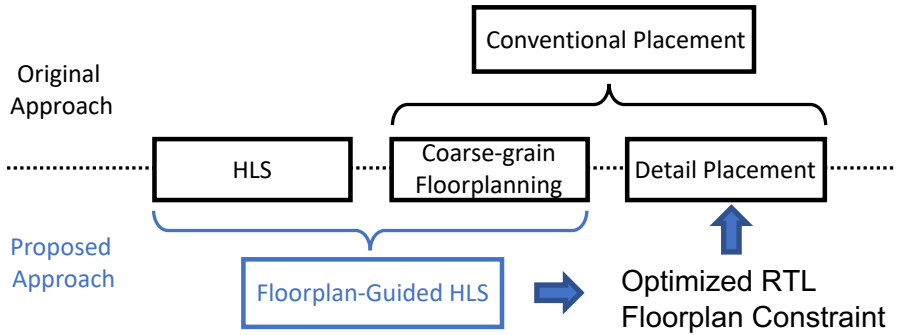
Core Idea

- Floorplan the design during HLS compilation
  - In a coarse granularity
- Add additional pipelining based on floorplan results
  - Guarantee no loss of performance

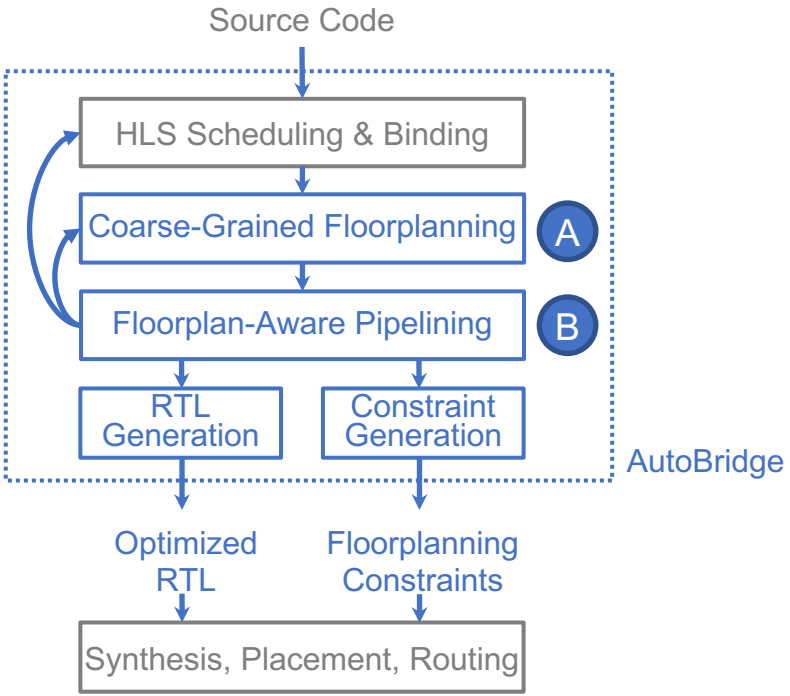


Core Idea

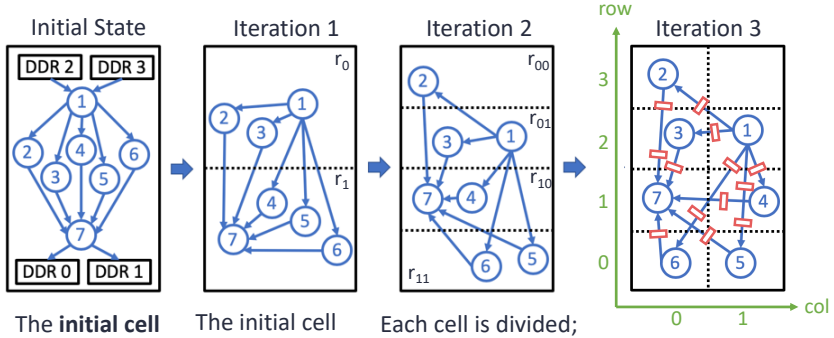
- Floorplan the design during HLS compilation
  - In a coarse granularity
- Add additional pipelining based on floorplan results
  - Guarantee no loss of performance



Framework Overview



A Integrate Top-Down Physical Planning with HLS



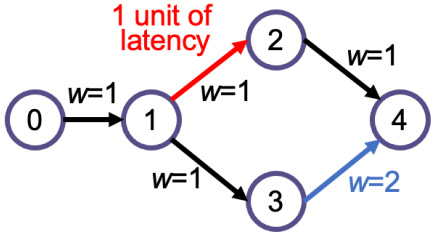
The **initial cell** representing the FPGA device

The initial cell is divided into two **child cells**.

Each cell is divided; r_0 divided into r_{00}, r_{01}; r_1 into r_{10}, r_{11}

Eventually form a 2x4 **grid** of cells

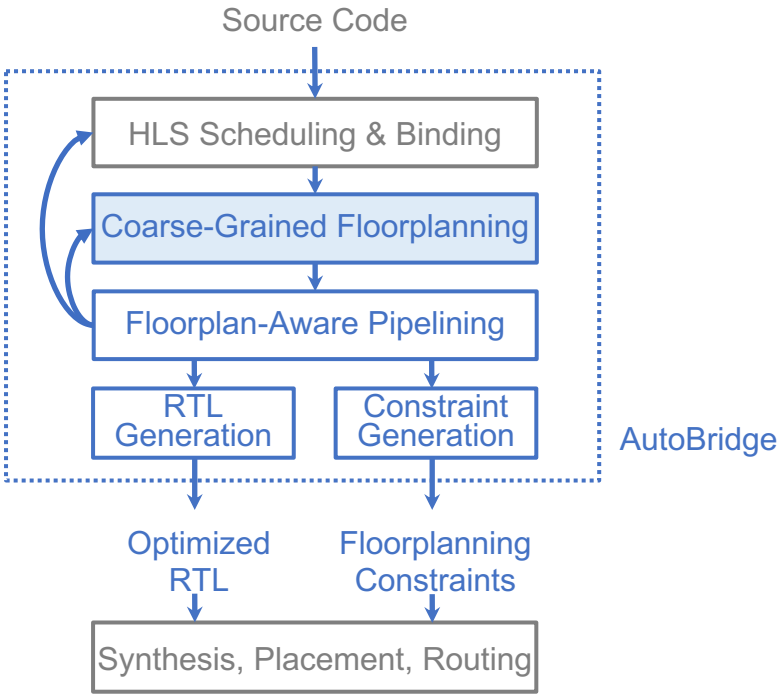
B Pipelining with Min. Area and Lossless Throughput



$$S_1 \geq S_2 + 1$$
$$S_2 \geq S_4$$
$$S_1 \geq S_3$$
$$S_3 \geq S_4$$
$$S_0 \geq S_1$$

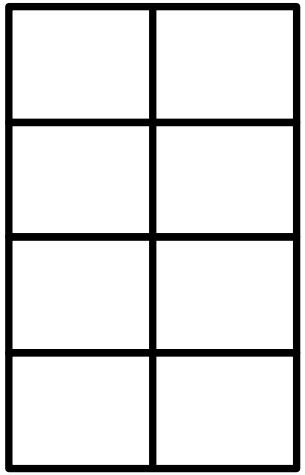
$$\text{min.} \quad (S_2 - S_4) + (S_1 - S_2) + (S_1 - S_3) + 2\,(S_3 - S_4) + (S_0 - S_1)$$

Framework Overview



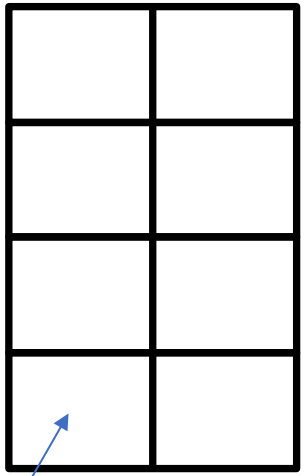
Coarse-Grained Floorplanning

- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot



Coarse-Grained Floorplanning

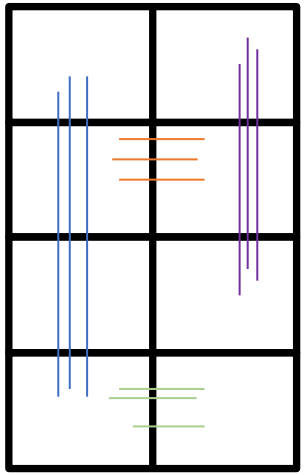
- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot
- Limit the resource utilization in each slot



limit resource usage (e.g., 70%)

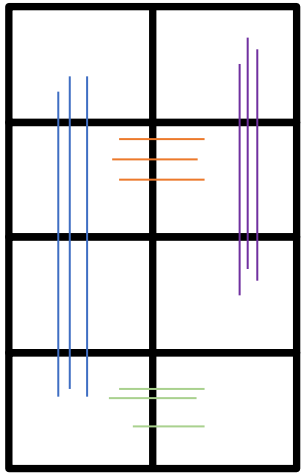
Coarse-Grained Floorplanning

- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot
- Limit the resource utilization in each slot
- Minimize the count of crossing-boundary wires



Coarse-Grained Floorplanning

- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot
- Limit the resource utilization in each slot
- Minimize the count of crossing-boundary wires
- It is OK to have ultra-long connections
    - Will be pipelined later



Coarse-Grained Floorplanning

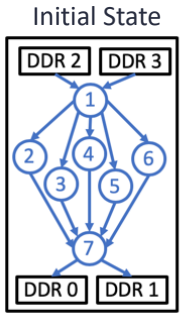
- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot
- Use ILP to iteratively partition the design

variables == # HLS functions

constraints == # connections

items in goal == # connections

Usual runtime < 10s



The **initial cell** representing the FPGA device

Coarse-Grained Floorplanning

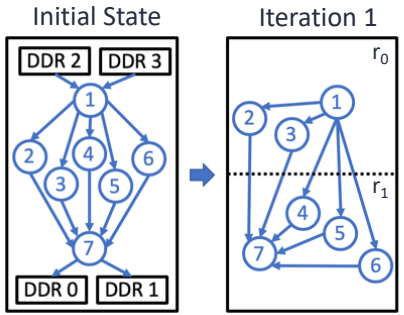
- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot
- Use ILP to iteratively partition the design

variables == # HLS functions

constraints == # connections

items in goal == # connections

Usual runtime < 10s



The **initial cell** representing the FPGA device

The initial cell is divided into two **child cells**.

Coarse-Grained Floorplanning

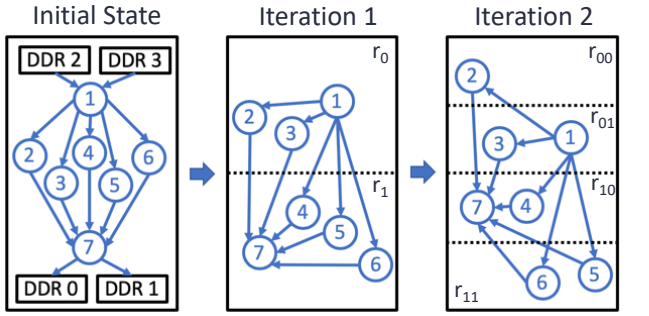
- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot
- Use ILP to iteratively partition the design

variables == # HLS functions

constraints == # connections

items in goal == # connections

Usual runtime < 10s



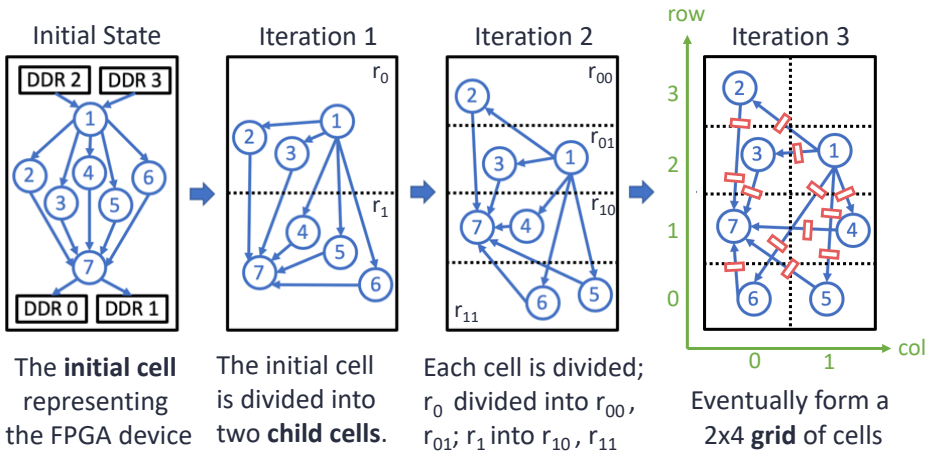
The **initial cell** representing the FPGA device

The initial cell is divided into two **child cells**.

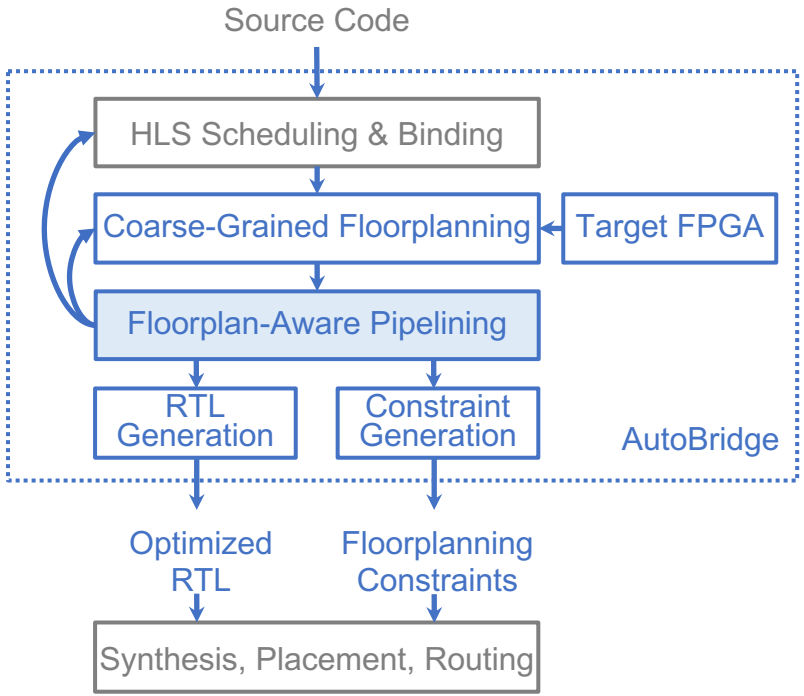
Each cell is divided; r_0 divided into r_{00}, r_{01}; r_1 into r_{10}, r_{11}

Coarse-Grained Floorplanning

- Divide the FPGA into a grid of slots
- Assign each HLS function to one slot
- Use ILP to iteratively partition the design
- Pipeline the cross-slot connections

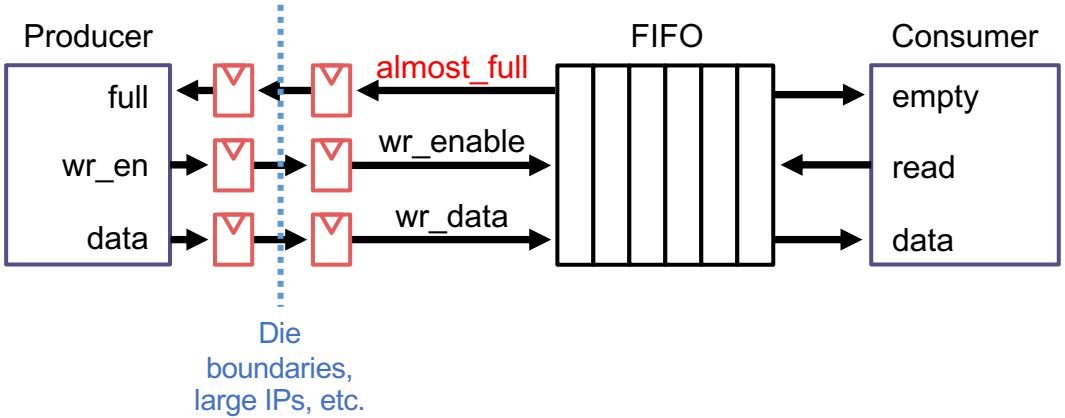


Framework Overview



Pipeline Data Transfer Logic

- We focus on flow-control interfaces (e.g., FIFO, AXI)
- Assume a dataflow programming model
- Can be extended to non-flow-control interface
  - Refer to our paper for details



Address the Performance Concern

- Focus on when modules communicate through FIFOs
  - Hard to statically analyze the impact of additional latency
  - The additional latency may cause throughput decrease

Address the Performance Concern

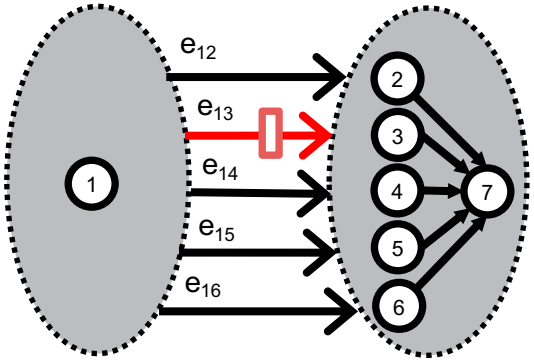
- Focus on when modules communicate through FIFOs
  - Hard to statically analyze the impact of additional latency
  - The additional latency may cause throughput decrease

Note that each FIFO is being accessed by an arbitrary function

$\Rightarrow$ Different from simplified model such as the Synchronous Data Flow (SDF)

Address the Performance Concern

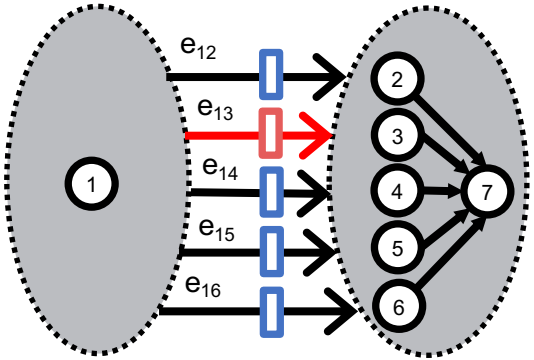
- Focus on when modules communicate through FIFOs
  - Hard to statically analyze the impact of additional latency
  - The additional latency may cause throughput decrease
- Adapt *cut-set pipelining*
  - Add the same latency to all edges in a cut
  - Equivalent to balancing the latency of reconvergent paths



Pipeline inter-slot connections

Address the Performance Concern

- Focus on when modules communicate through FIFOs
  - Hard to statically analyze the impact of additional latency
  - The additional latency may cause throughput decrease
- Adapt *cut-set pipelining*
  - Add the same latency to all edges in a cut
  - Equivalent to balancing the latency of reconvergent paths

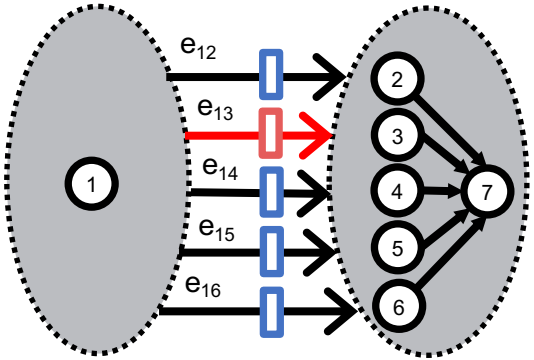


Pipeline inter-slot connections

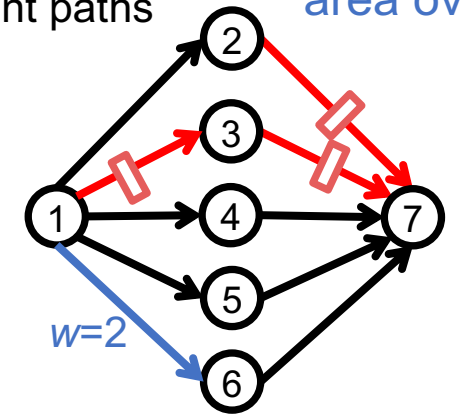
Balance the latency of all paths

Address the Performance Concern

- Focus on when modules communicate through FIFOs
  - Hard to statically analyze the impact of additional latency
  - The additional latency may cause throughput decrease
- Adapt *cut-set pipelining*
  - Add the same latency to all edges in a cut
  - Equivalent to balancing the latency of reconvergent paths

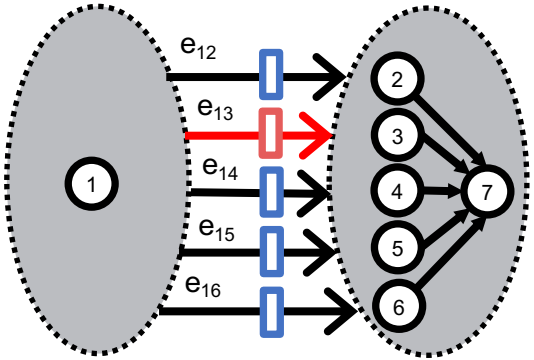


How to minimize area overhead?

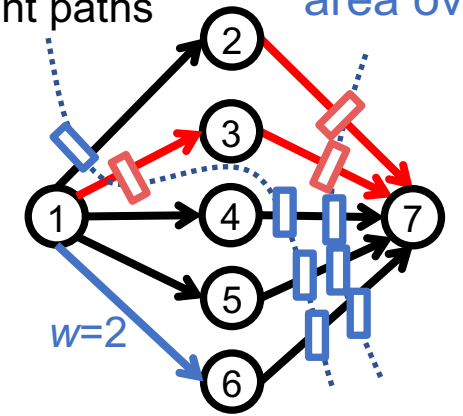


Address the Performance Concern

- Focus on when modules communicate through FIFOs
  - Hard to statically analyze the impact of additional latency
  - The additional latency may cause throughput decrease
- Adapt *cut-set pipelining*
  - Add the same latency to all edges in a cut
  - Equivalent to balancing the latency of reconvergent paths



How to minimize area overhead?



Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable S_v for each vertex v
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x – S_y)$ represents the latency of all path between vertex x and y

Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable S_v for each vertex v
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x – S_y)$ represents the latency of all path between vertex x and y
- For an edge e_{uv} , $(S_u – S_v)$ is no less than the additional latency needed for this edge

Latency Balancing with Minimal Area Overhead

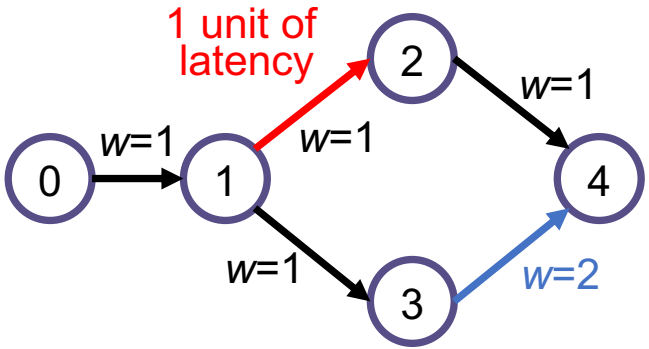
Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable S_v for each vertex v
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x – S_y)$ represents the latency of all path between vertex x and y
- For an edge e_{uv} , $(S_u – S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead

Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable S_v for each vertex v
  - Analogous to the "arrival time" in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex x and y
- For an edge e_{uv}, $(S_u - S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead



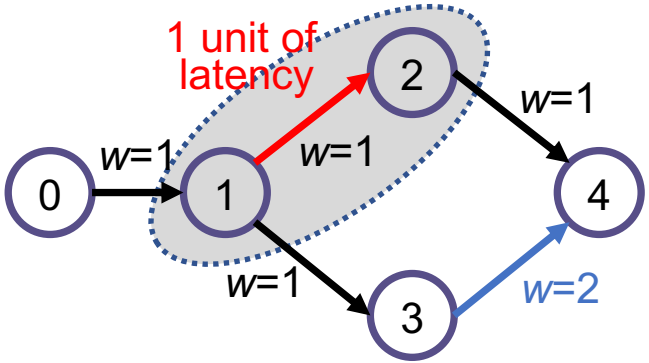
$$S_1 \geq S_2 + 1$$
$$S_2 \geq S_4$$
$$S_1 \geq S_3$$
$$S_3 \geq S_4$$
$$S_0 \geq S_1$$

$$\text{min.} \quad (S_2 - S_4) + (S_1 - S_2) + (S_1 - S_3) + 2\,(S_3 - S_4) + (S_0 - S_1)$$

Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable S_v for each vertex v
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x – S_y)$ represents the latency of all path between vertex x and y
- For an edge e_{uv}, $(S_u – S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead

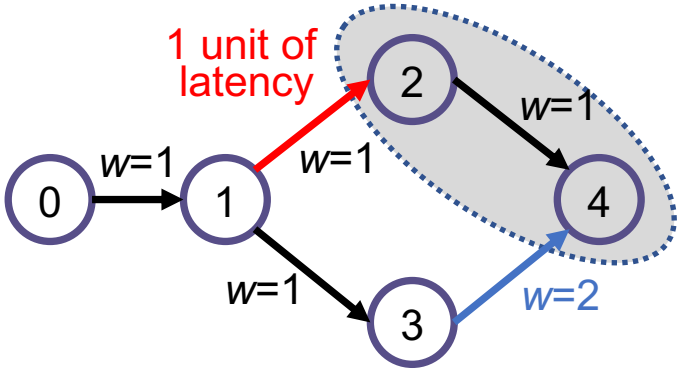


$S_1 \geq S_2 + 1$

Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable S_v for each vertex v
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x – S_y)$ represents the latency of all path between vertex x and y
- For an edge e_{uv}, $(S_u – S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead

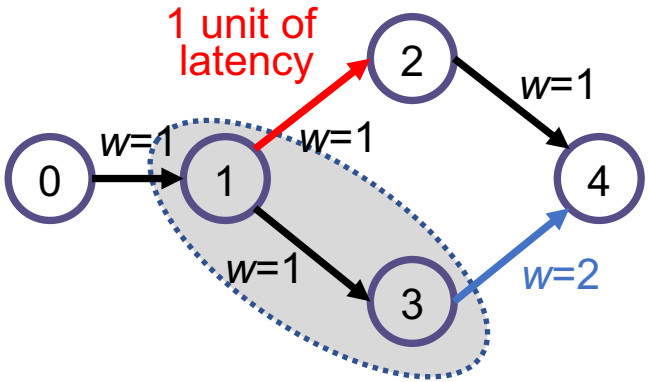


$S_1 \geq S_2 + 1$
$S_2 \geq S_4$

Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable S_v for each vertex v
  - Analogous to the "arrival time" in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex x and y
- For an edge e_{uv}, $(S_u - S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead

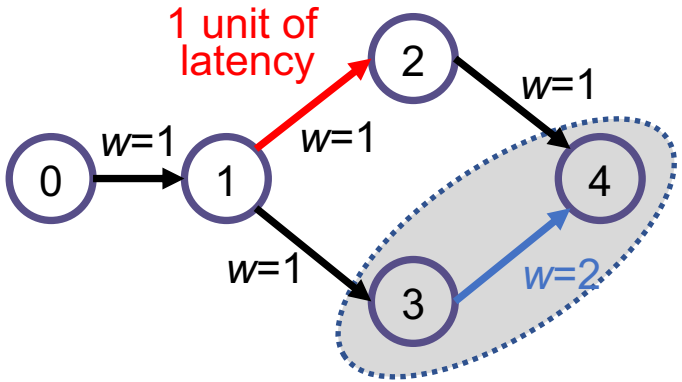


$S_1 \geq S_2 + 1$
$S_2 \geq S_4$
$S_1 \geq S_3$

Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable S_v for each vertex v
  - Analogous to the “arrival time” in static timing analysis
  - $(S_x – S_y)$ represents the latency of all path between vertex x and y
- For an edge e_{uv}, $(S_u – S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead

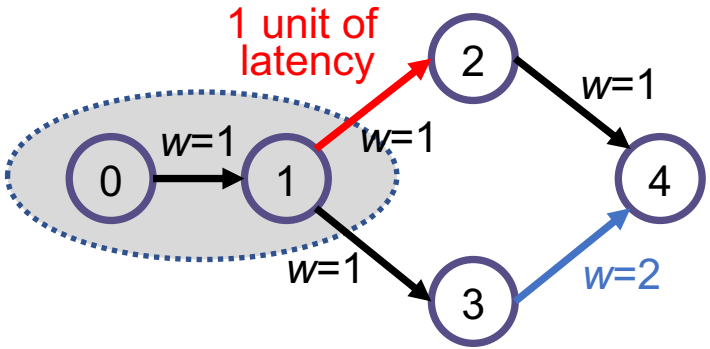


$S_1 \geq S_2 + 1$
$S_2 \geq S_4$
$S_1 \geq S_3$
$S_3 \geq S_4$

Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable S_v for each vertex v
  - Analogous to the "arrival time" in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex x and y
- For an edge e_{uv}, $(S_u - S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead

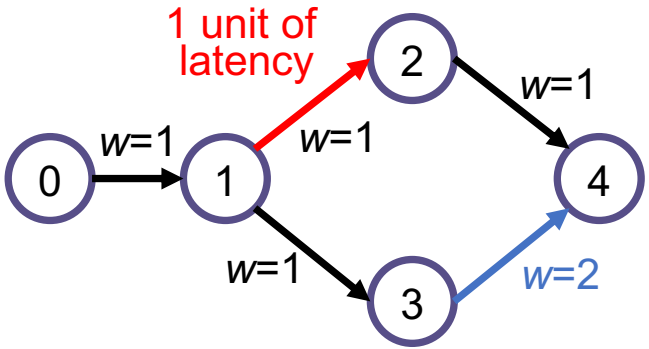


$S_1 \geq S_2 + 1$
$S_2 \geq S_4$
$S_1 \geq S_3$
$S_3 \geq S_4$
$S_0 \geq S_1$

Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable S_v for each vertex v
  - Analogous to the "arrival time" in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex x and y
- For an edge e_{uv}, $(S_u - S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead



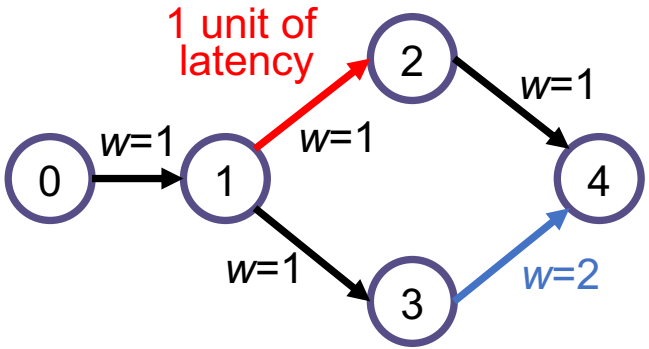
$$S_1 \geq S_2 + 1$$
$$S_2 \geq S_4$$
$$S_1 \geq S_3$$
$$S_3 \geq S_4$$
$$S_0 \geq S_1$$

$$\text{min.} \quad (S_2 - S_4) + (S_1 - S_2) + (S_1 - S_3) + 2\,(S_3 - S_4) + (S_0 - S_1)$$

Latency Balancing with Minimal Area Overhead

Problem: balance the latency of every pair of reconvergent paths with min area.

- Assign variable S_v for each vertex v
  - Analogous to the "arrival time" in static timing analysis
  - $(S_x - S_y)$ represents the latency of all path between vertex x and y
- For an edge e_{uv}, $(S_u - S_v)$ is no less than the additional latency needed for this edge
- Minimize the area overhead

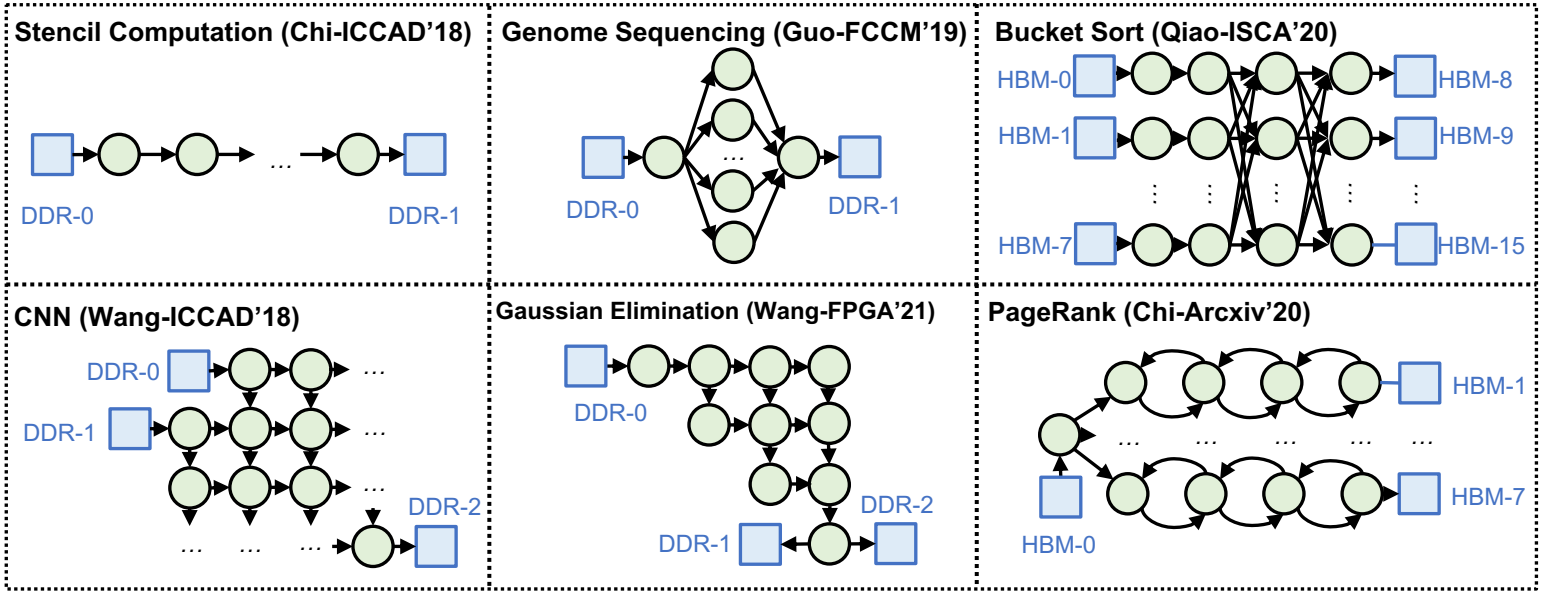


$$S_1 \geq S_2 + 1$$
$$S_2 \geq S_4$$
$$S_1 \geq S_3$$
$$S_3 \geq S_4$$
$$S_0 \geq S_1$$

System of Difference Constraints (Polynomial Time Solvable)

$$\text{min.} \quad (S_2 - S_4) + (S_1 - S_2) + (S_1 - S_3) + 2\,(S_3 - S_4) + (S_0 - S_1)$$

Benchmarks



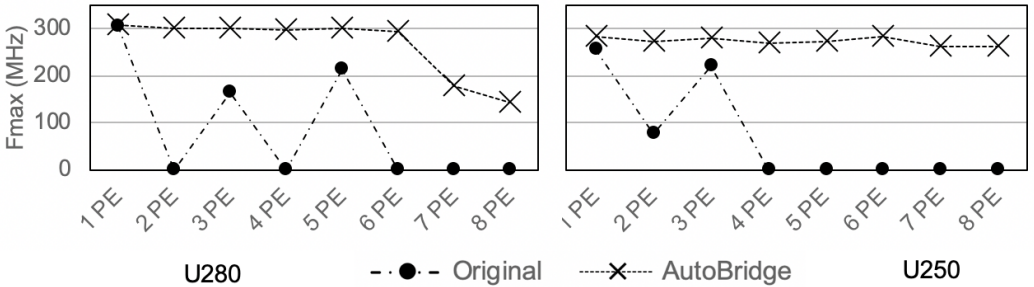
- A total of 43 design configurations
- 16 of them originally failed in routing
- From 147 MHz to 297 MHz on average (~2X)
- Negligible difference in resource utilization or cycle count.

Case Study 1

- Stencil Computation, 16 configurations

Opt: avg 266 MHz (3.1X)

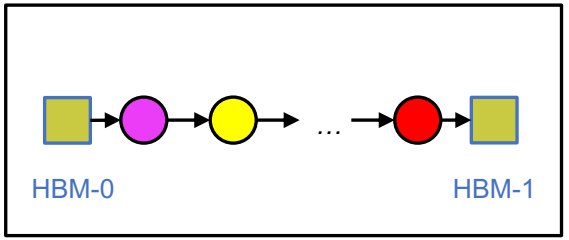
Opt: avg. 273 MHz (3.9X)

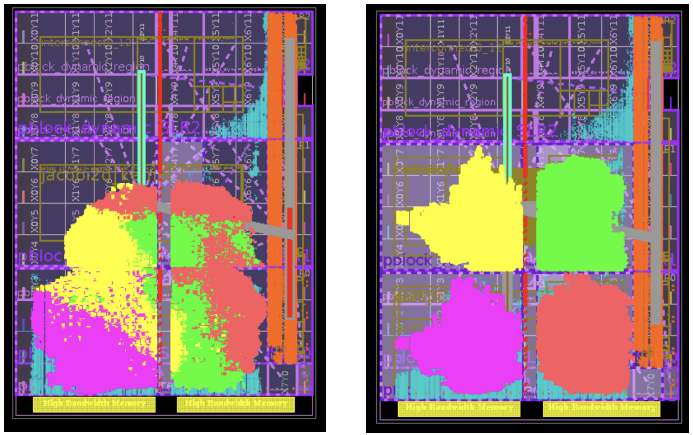


Default: avg. 86 MHz

Default: avg. 69 MHz

- Difference in Resource Utilization
  - LUT: -0.26%
  - FF: +0.78%
  - BRAM: +4.68%
  - DSP: +0.00%





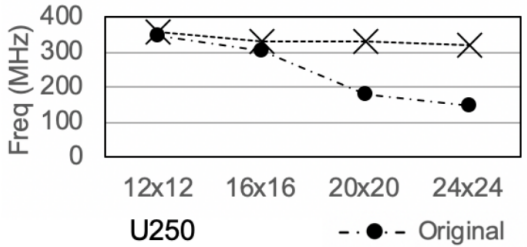
Default AutoBridge

Comparison of the 4-PE Design on U280

Case Study 2

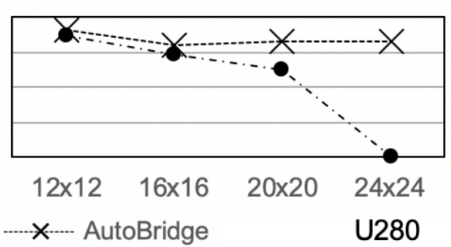
- Gaussian Elimination, 8 configurations

Opt: avg. 334 MHz (1.4X)



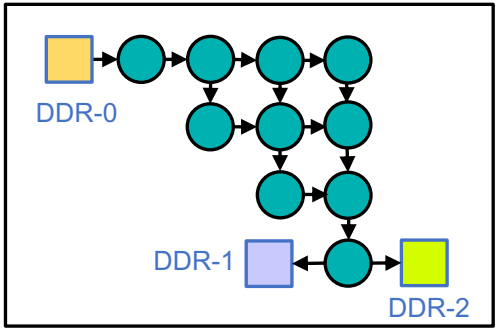
Default: avg. 245 MHz

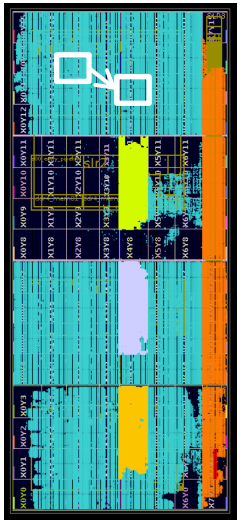
Opt: avg. 335 MHz (1.5X)

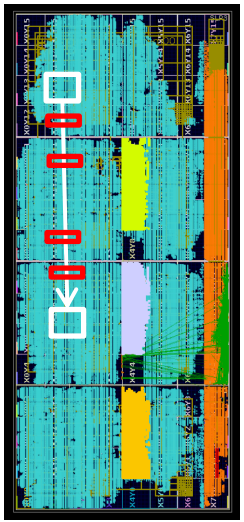


Default: avg. 223 MHz

- Difference in Resource Utilization
  - LUT: -0.14%
  - FF: -0.04%
  - BRAM: -0.03%
  - DSP: +0.00%





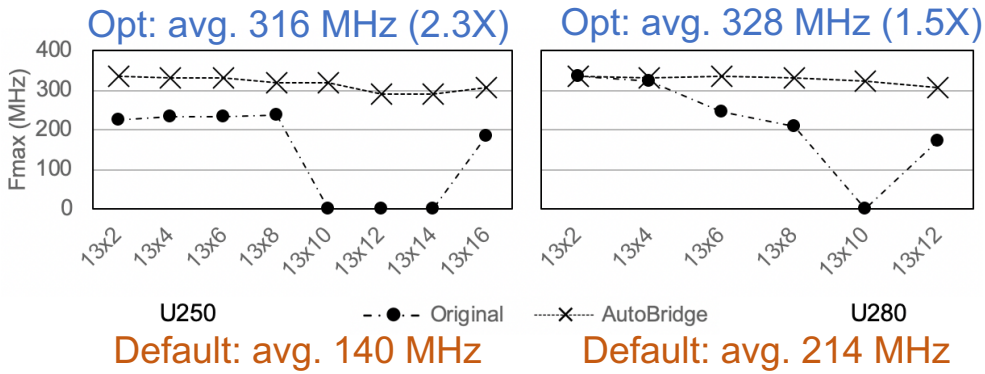


Default AutoBridge

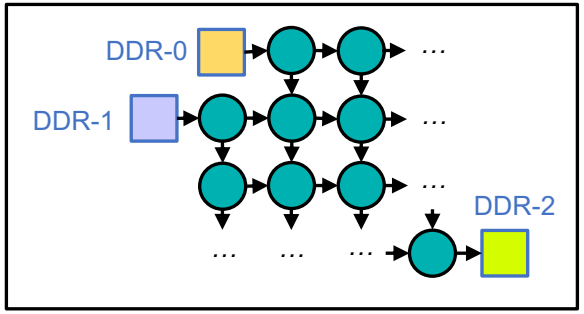
Comparison of the 24x24 Design on U250

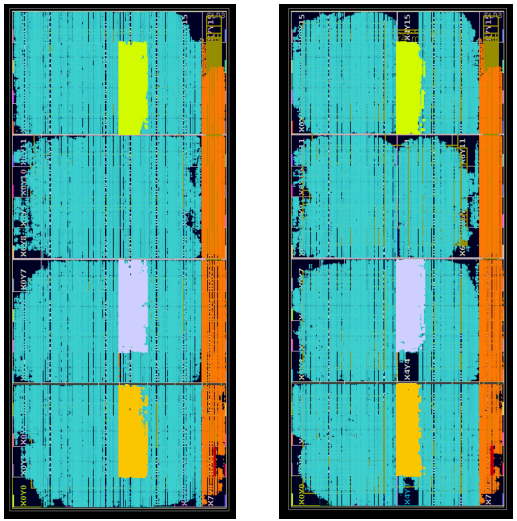
Case Study 3

- CNN Accelerator, 14 configurations



- Difference in Resource Utilization
  - LUT: -0.08%
  - FF: -0.16%
  - BRAM: -0.02%
  - DSP: +0.00%



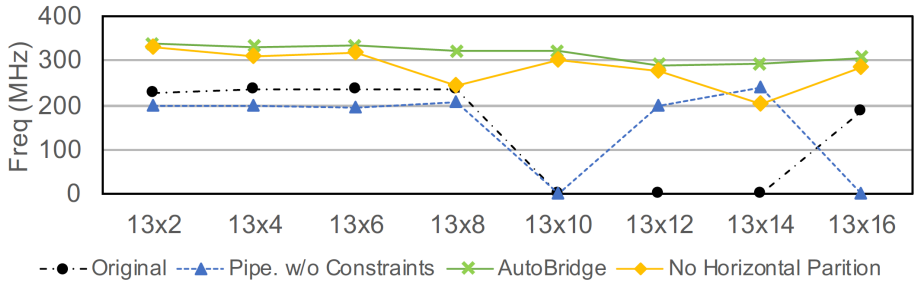


Default AutoBridge

Comparison of the 16x13 Design on U250

Impact of Pipelining and Floorplanning

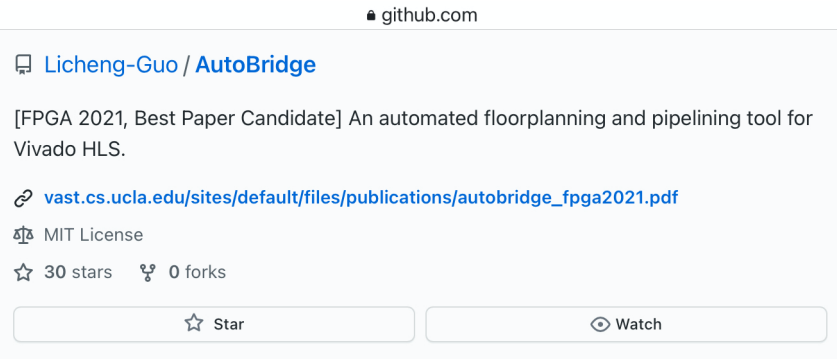
- Is it possible that only one of them is the key factor?
  - Baseline: (-) floorplanning, 8 slots (-) pipelining
  - AutoBridge: (+) floorplanning, 8 slots (-) pipelining
  - Case 1: (-) floorplanning (+) pipelining
  - Case 2: (+) floorplanning, 4 slots (neglect the DDRs) (-) pipelining



Control Experiments Based on Systolic Arrays on U250

Projects Using AutoBridge

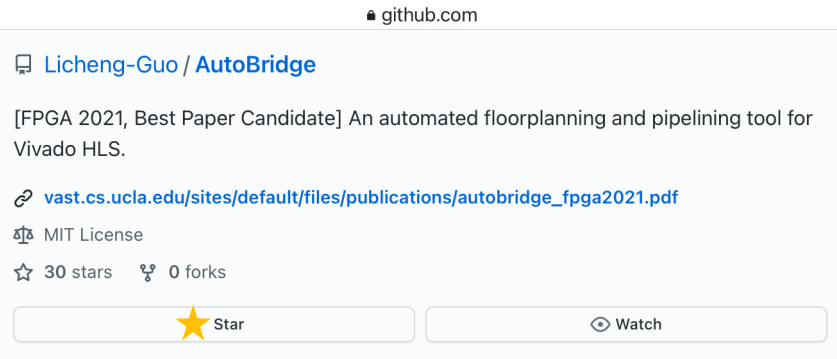
- AutoSA: Polyhedral-Based Systolic Array Auto-Compilation
  - https://github.com/UCLA-VAST/AutoSA
- TAPA: Extending High-Level Synthesis for Task-Parallel Programs
  - https://github.com/Blaok/tapa
- Acceleration of Bayesian Network Inference (in submission)
- Acceleration of Single-Source-Shortest-Path algorithm (in submission)



https://github.com/Licheng-Guo/AutoBridge

Projects Using AutoBridge

- AutoSA: Polyhedral-Based Systolic Array Auto-Compilation
  - https://github.com/UCLA-VAST/AutoSA
- TAPA: Extending High-Level Synthesis for Task-Parallel Programs
  - https://github.com/Blaok/tapa
- Acceleration of Bayesian Network Inference (in submission)
- Acceleration of Single-Source-Shortest-Path algorithm (in submission)



https://github.com/Licheng-Guo/AutoBridge

Thank You!