IBM Austin

# Quantifying Academic Placer Performance on Custom Designs

## Datapath Placement Benchmarks

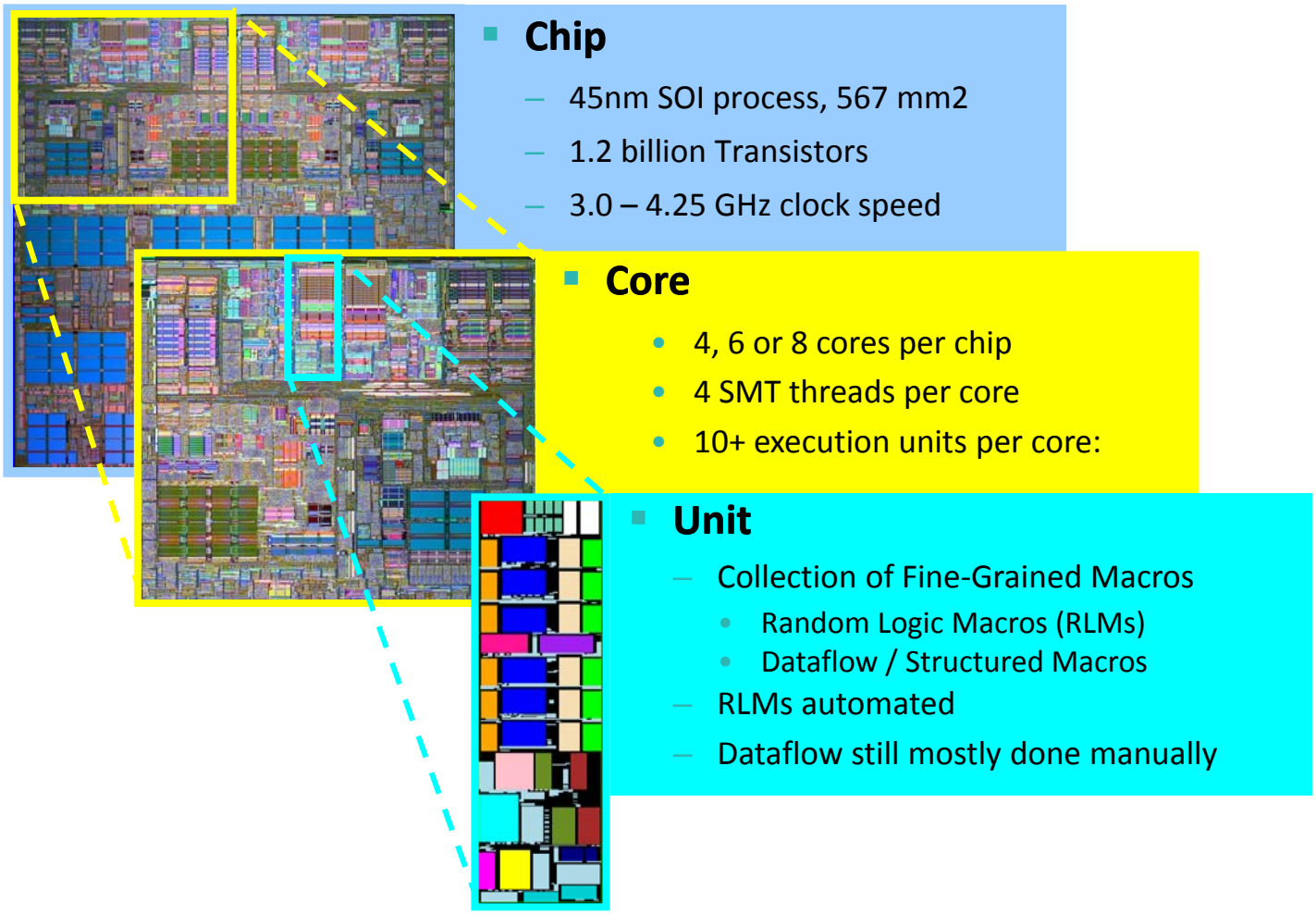
Samuel Ward and Earl Swartzlander — The University of Texas at Austin

David A. Papa, Zhuo Li,
Cliff Sze, and Charles Alpert — IBM Austin Research

# Outline

- **Background**
- **Motivation**
- **What is Dataflow or Structured Placement Design?**
- **Dataflow Circuit Design Examples**
- **Structured Placement Benchmark A**
- **Structured Placement Benchmark B**
- **Errata to the Paper**
- **Results**

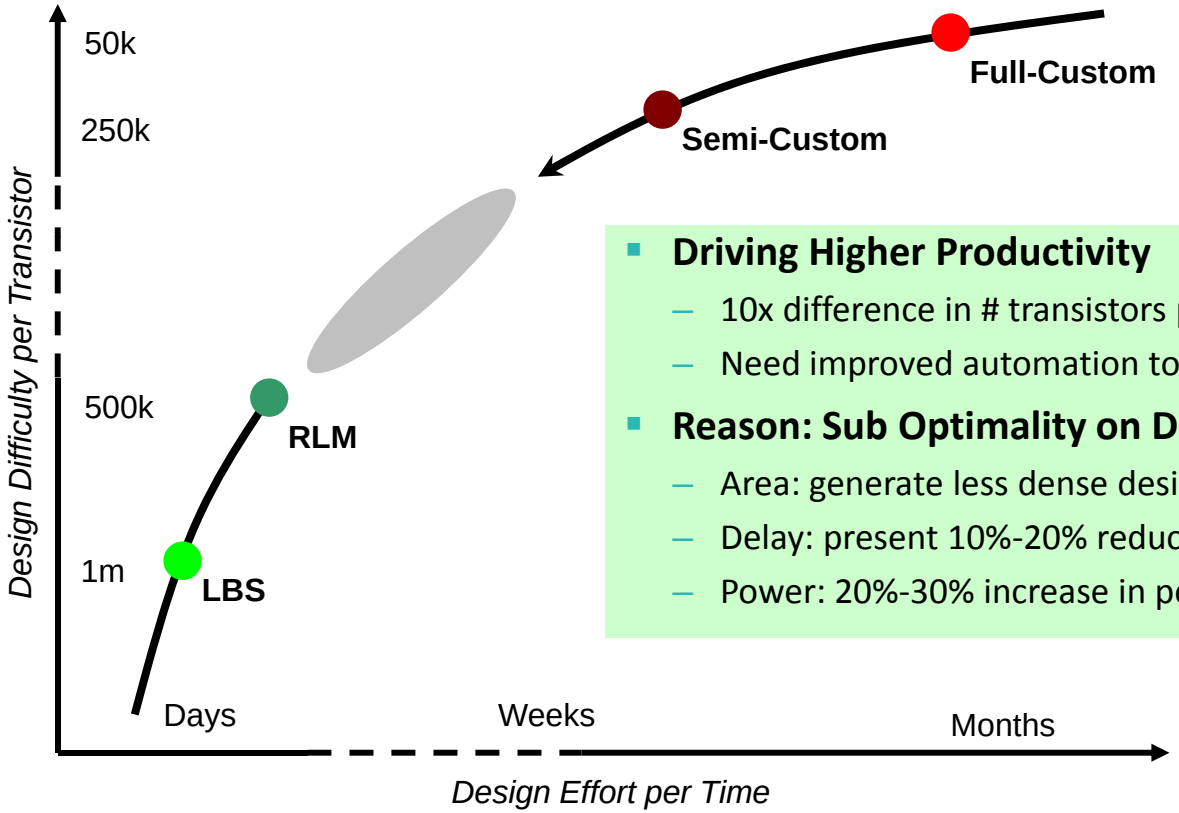
# Modern Processor Design – Power7



- **Chip**
  - 45nm SOI process, 567 mm2
  - 1.2 billion Transistors
  - 3.0 – 4.25 GHz clock speed
- **Core**
  - 4, 6 or 8 cores per chip
  - 4 SMT threads per core
  - 10+ execution units per core:
- **Unit**
  - Collection of Fine-Grained Macros
    - Random Logic Macros (RLMs)
    - Dataflow / Structured Macros
  - RLMs automated
  - Dataflow still mostly done manually

# Motivation

- **Bridging the Gap:**




- **Driving Higher Productivity**
  - 10x difference in # transistors per designer for RLM vs Dataflow
  - Need improved automation tools because of growing costs
- **Reason: Sub Optimality on Dataflow Macros**
  - Area: generate less dense designs
  - Delay: present 10%-20% reduction post routed delays
  - Power: 20%-30% increase in power for similar delays

# Goals

- **Often claimed that current placement tools do not perform well on datapath designs**
- **BUT WHY?? AND HOW MUCH?? Is it the…**
  - Regular structure?
  - Highly compact layouts?
- **<u>GOAL:</u> Compare State-of-the-art placement tools against manual placement on REAL designs**
  - Is placement the problem or is it something else?
- **This work:**
  - Presents two custom constructed datapath designs that perform common logic functions
  - Presents hand-designed layouts for each to compare the “known optimal solution”
  - Compares latest generation of academic placers against them
  - Testcases released publically

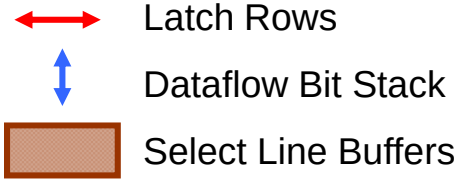
# What's Different?

- **There have been past attempts to quantify suboptimality of placement heuristics**
  - Hagen, *et al.* [7]
    - Copy small circuits and replicate them
    - Loosely connecting their ports together, in order to create a much larger benchmark.
    - Problems
      - Defined connections between the copies do not correspond to real logic functions.
      - No pin locations are defined for the circuit
  - PEKO/PEKU Benchmarks - Chang, *et al.*
    - Placement examples with known optima (PEKO)
    - Placement examples with known upperbounds (PEKU)
    - Optimality achieved by adding nets to cells in configurations that cannot be shortened.
    - though the pin distributions of cells matched that of a typical VLSI circuit.
    - Problems
      - these netlists did not correspond to any logic function at all.
      - It could be argued that the PEKO and PEKU testcases are artificially hard
- **What is Needed:**
  - Need real logic function
  - Need to now how close placers are to optimal

# What is Dataflow Design?

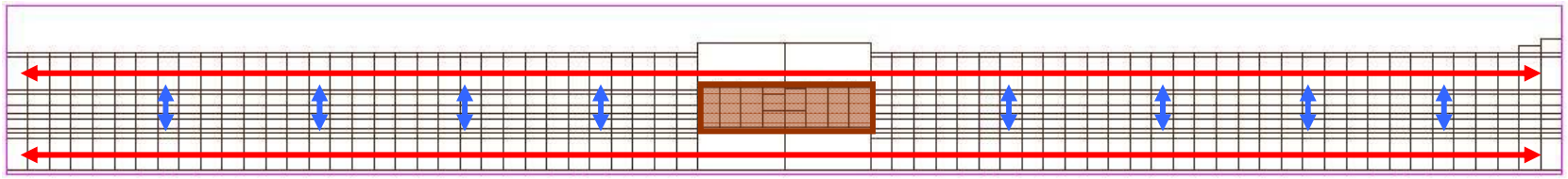
- **Logic Function Types:**
  - Load / Store queue
  - Decoders
  - Encoders
  - Crossbar Switch
  - Adders
  - Muxes
  - Latch Banks for Buses
  - Memories / CAMs
- Many designs have regular datapaths, placers have no regular structure
- Placement failures: high utilization, irregular shapes
- Gate Sizing: Larger wire lengths cause increased gate sizes
- Routing: Difficult to route even they are placed
- Let's look at some examples…

# Dataflow Example 1: Fixed Latches

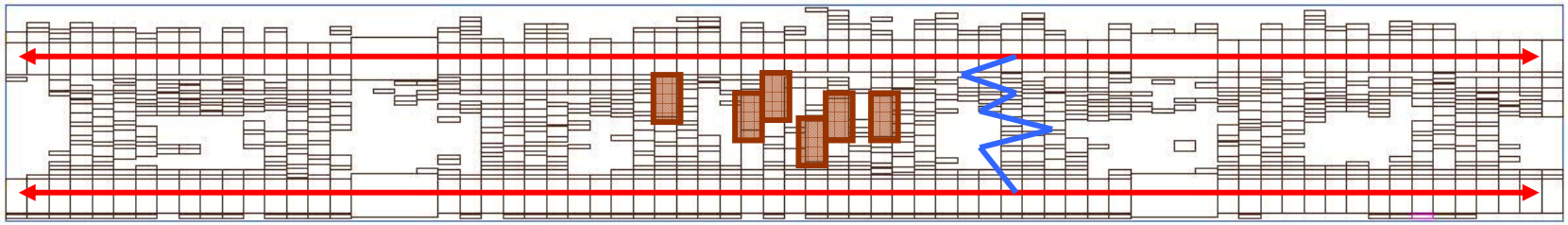



- **Select Lines Share Routing Tracks**
- **Bitstack Compressed to Save Area / Routing**
- **>30% Area Growth**
- **Input / Output Pins Lined up to Reduce Integration Level Congestion**

- **Custom Solution:**

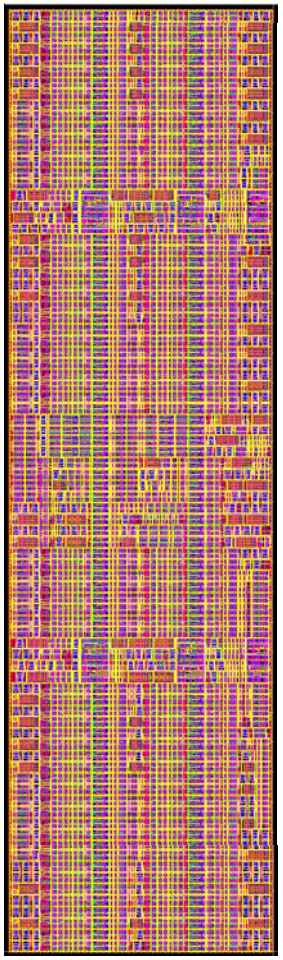


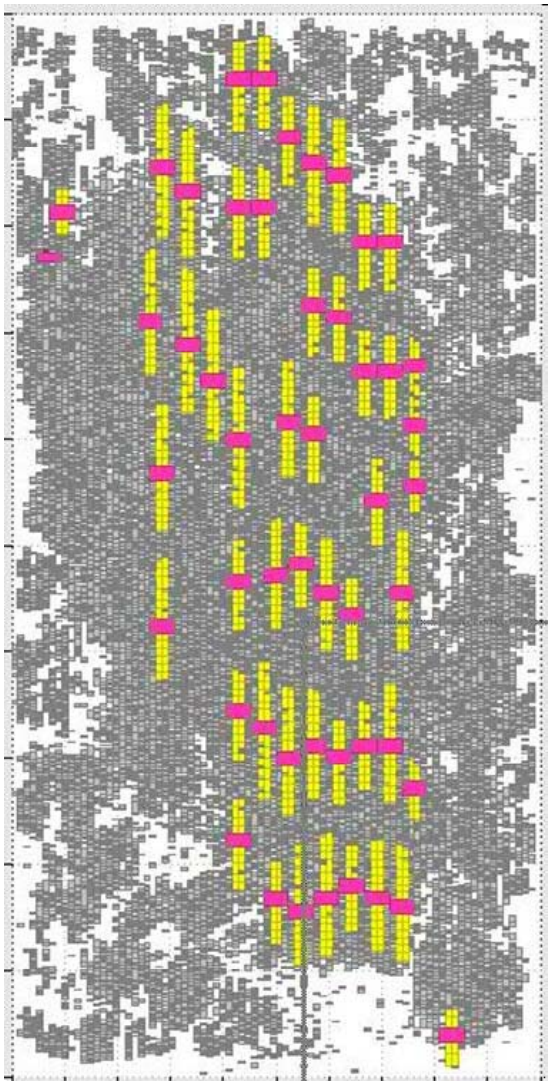
- **Placed Solution:**



# Dataflow Example 2: Unfixed Latches

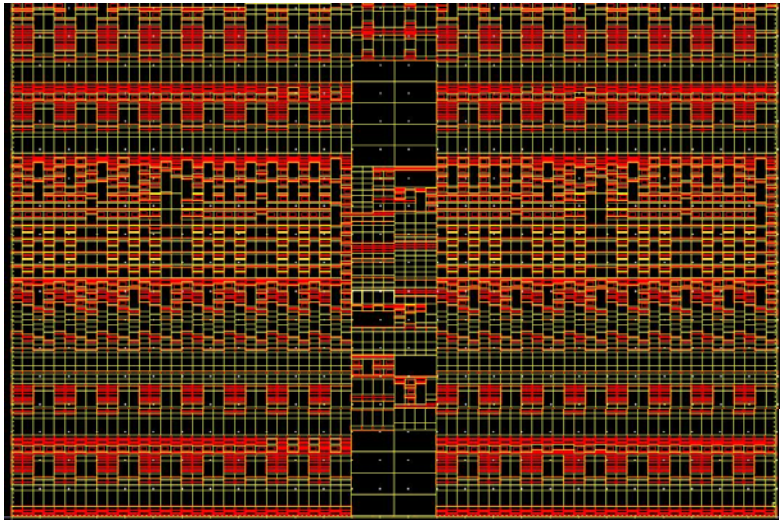
- **Impact**
  - To achieve similar wire length:
  - Area: >40%
  - Timing: >20%
- **Why Fixed Latches?**
  - Provide “hints” to the placer for improved results
  - Improve clock routing
  - Overall timing is better
  - Unfixed more Unstable -> multiple tool flows

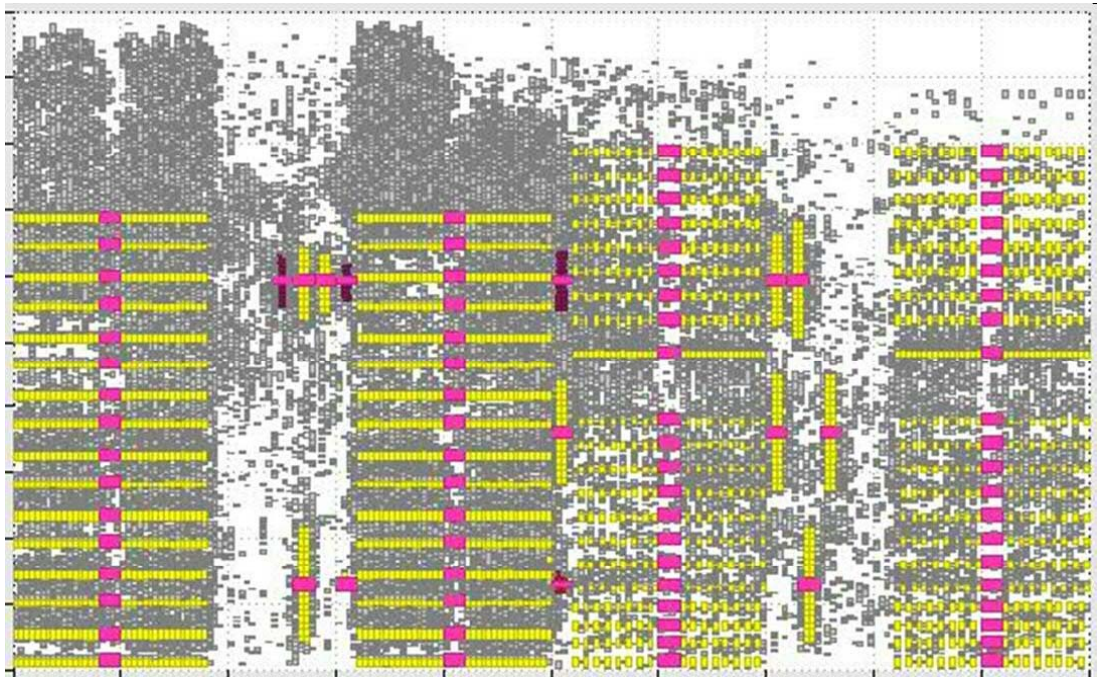




# Dataflow Examples: Design 3

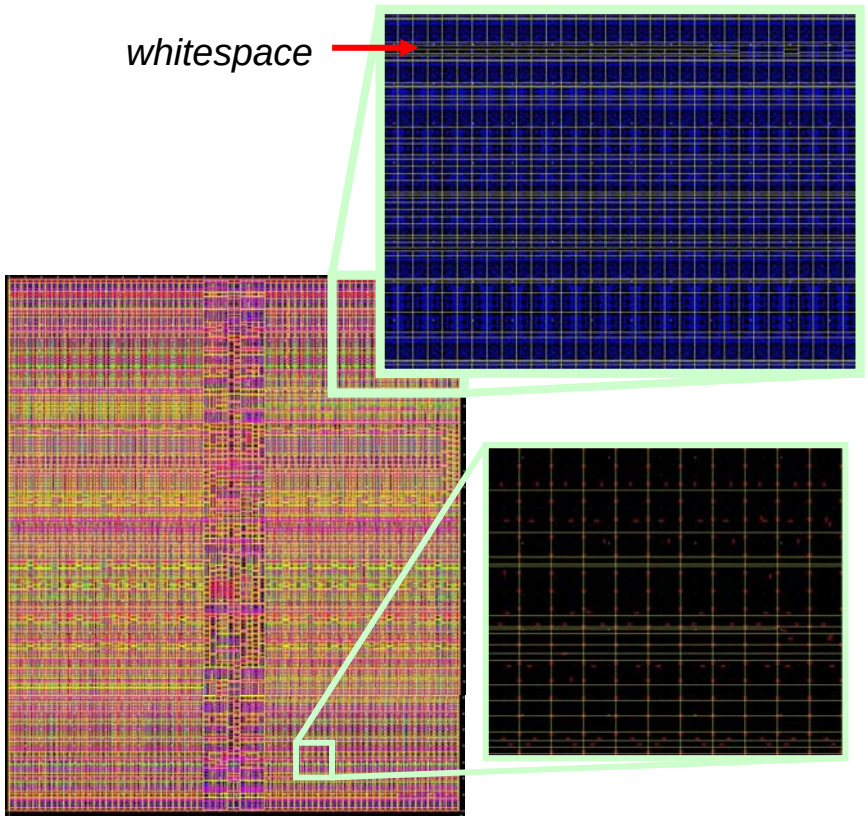
- **Custom Placement**
  - Util = 95%
  - Meets Timing
  - Design Time: 14-18 weeks
  - Highly Stable
- **Automated Placement**
  - Util < 70%
  - Routing Congestion Problems
  - Larger WL drives more power, larger gate sizes
  - ~40% Area Growth
  - Design Time: 6-8 weeks



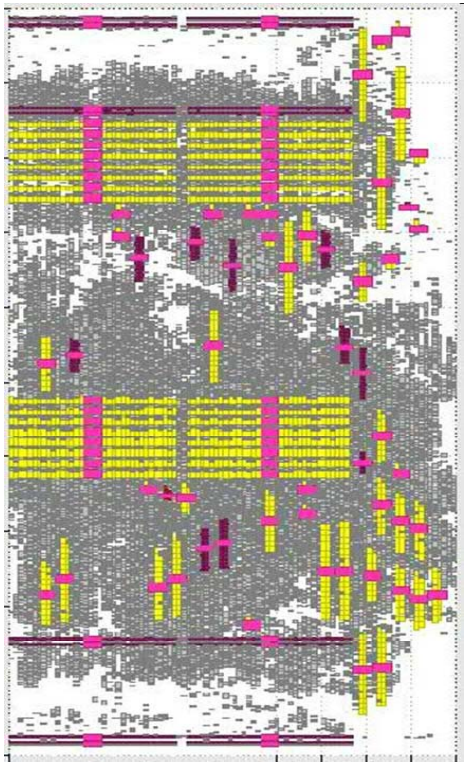


# Dataflow Examples: Design 4

- **Custom Placement**
  - Util > 95%, Meets Timing
  - Compact Placement, careful whitespace usage
  - Design Time: 8-12 Weeks

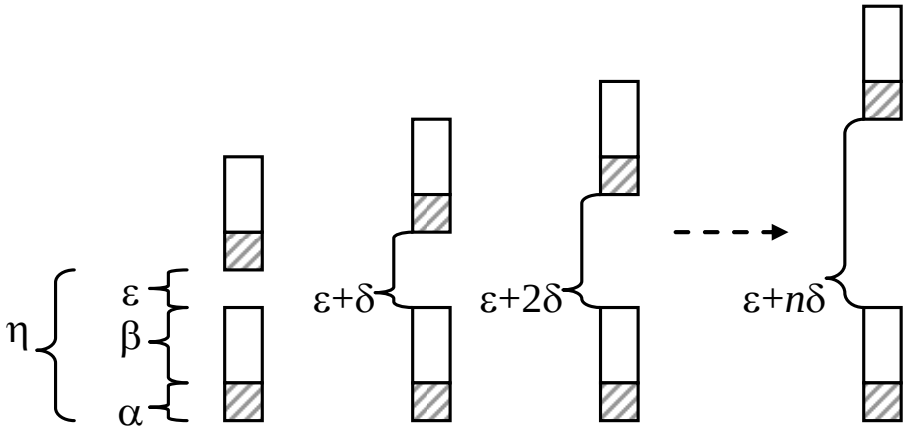



- **Automated Placement**
  - Util < 70%, Timing Critical
  - Larger WL drives more power, larger GW
  - ~30% Area Growth
  - Design Time: 2-4 Weeks



# Why are Placers Bad?

- **Generally speculated that poor performance of placers on datapath designs is due to very tight density constraints**
- **Perhaps placers could find the right structures but simply had trouble with the legalization?**
- **Experiment:**
  - Two dataflow designs built
  - Eight variants of each created
    - Additional whitespace inserted
    - Provides more opportunity for the placers




Total Cell Height: $\eta = \alpha + \beta + n\varepsilon$

# How Were They Built?

- **Generally, Custom Design uses a Different Library**
- **Implemented Common Dataflow Structures**
  - Custom Design Environment
  - Used Standard Cell Library
- **Manually Placed Custom**
- **Converted Layout Netlist to Bookshelf Format**
- **Compared Wire Length between Manual and Placed Solutions**
- **Let's take a look at the designs…**

# Design 1: Rotator (Barrel Shifter)

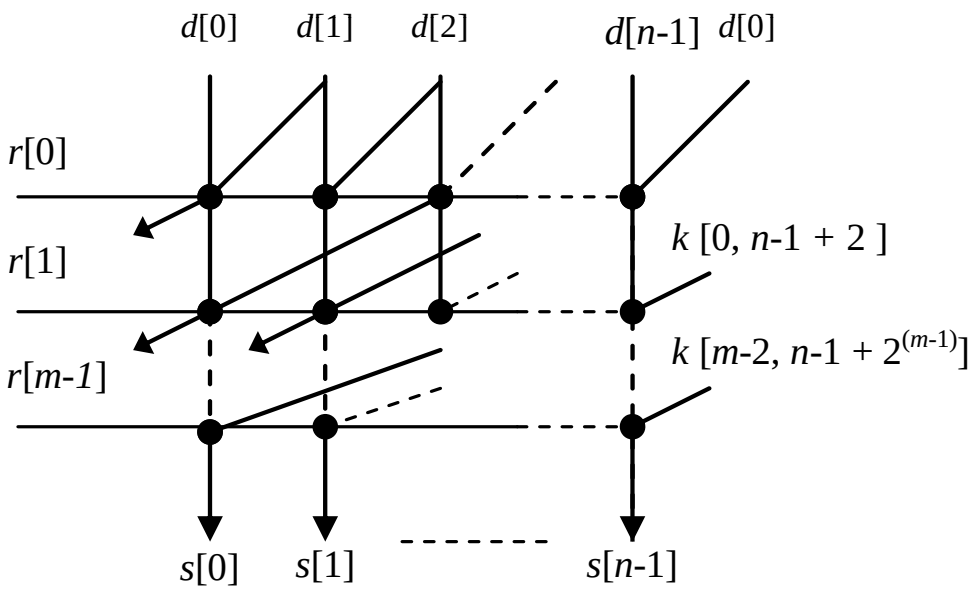
- **Rotate circuits**
  - Also known as cyclic shifters
  - A simple and common bit operation
- **Found throughout**
  - microprocessors,
  - cryptography,
  - imaging,
  - biometrics
- **Traditionally, custom designed because of**
  - Highly regular structure
  - Significant routing complexity (local and global)

# Design 1: Logical Overview

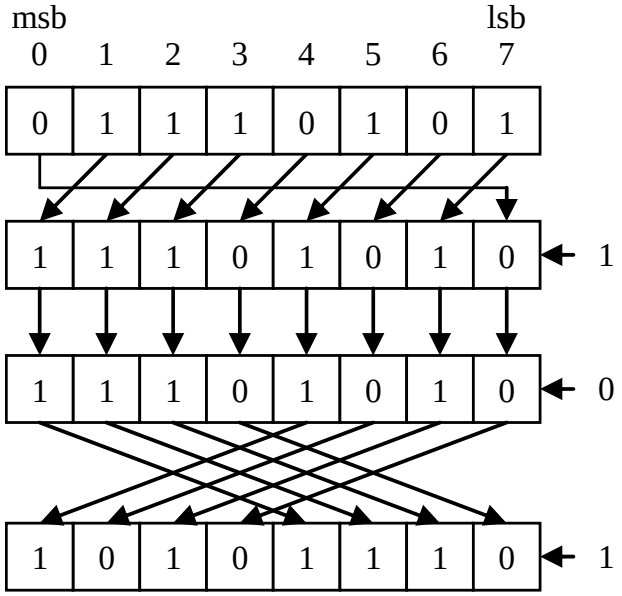
$k[i,j] = r[i]\ \&\ k[i\text{-}1\ ,\ j] + !\ r[i]\ \&\ k[\ i\text{-}1\ ,\ j + 2i]$

where i = 0, ..., $m$-1 , j = 1, ..., n-1

$k[i, j] = k[i, j + z * n]$, where $z$ is 0, 1, 2, ...,

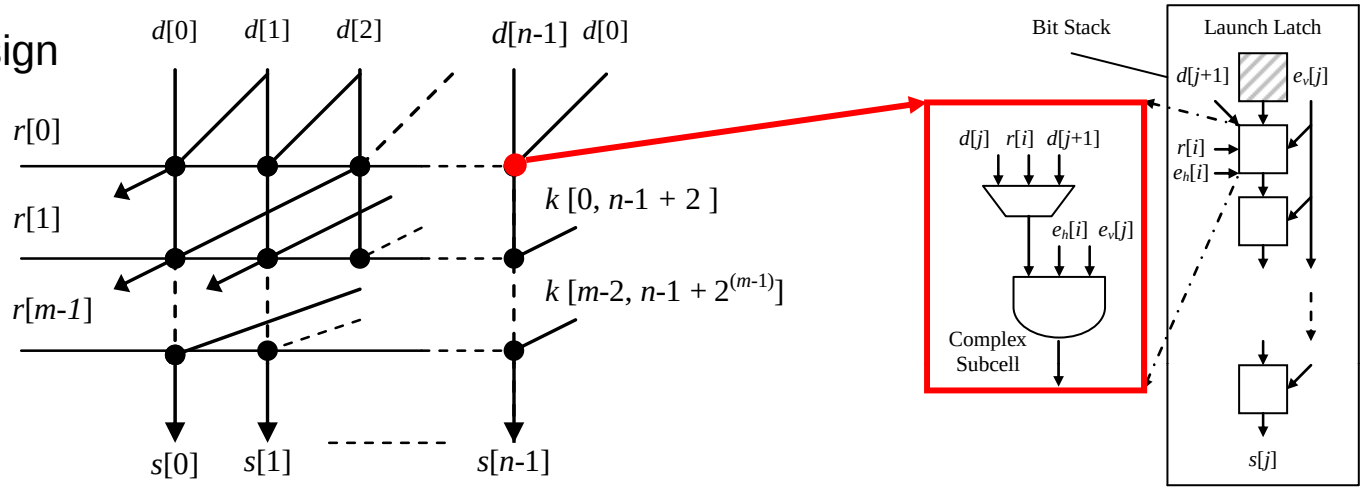



- **Example:**
  - d:01110101
  - R:101
  - S:10101110

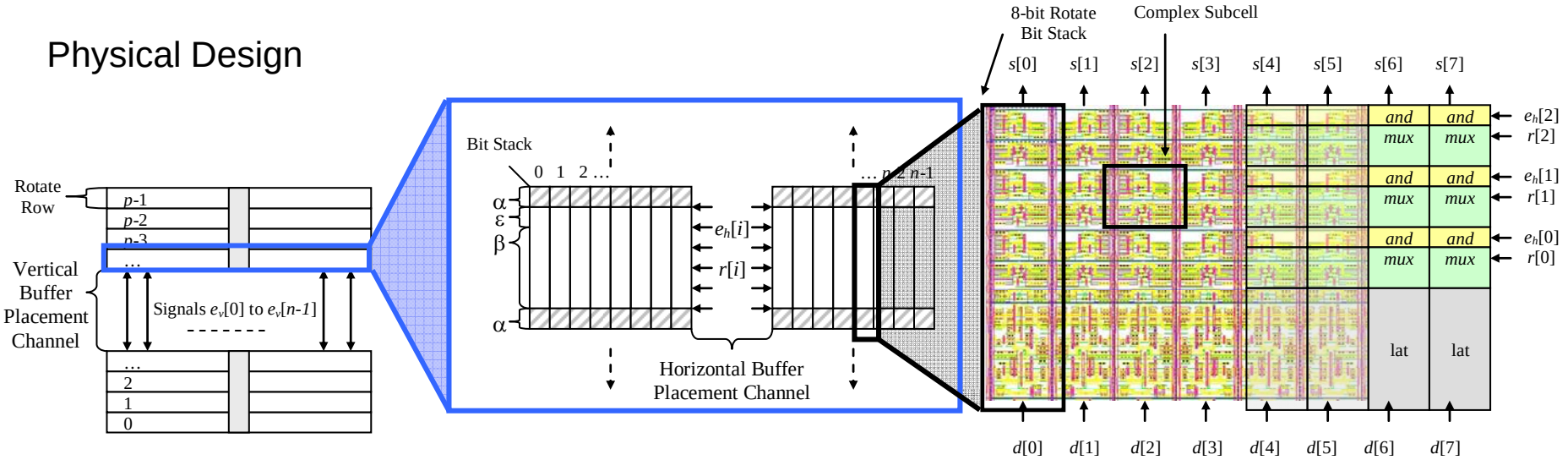


# Design 1 Placement

## Logical Design

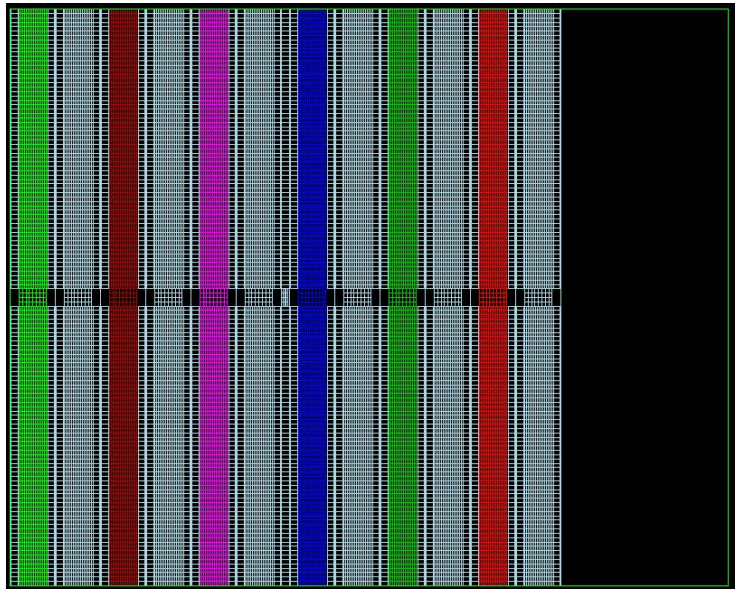


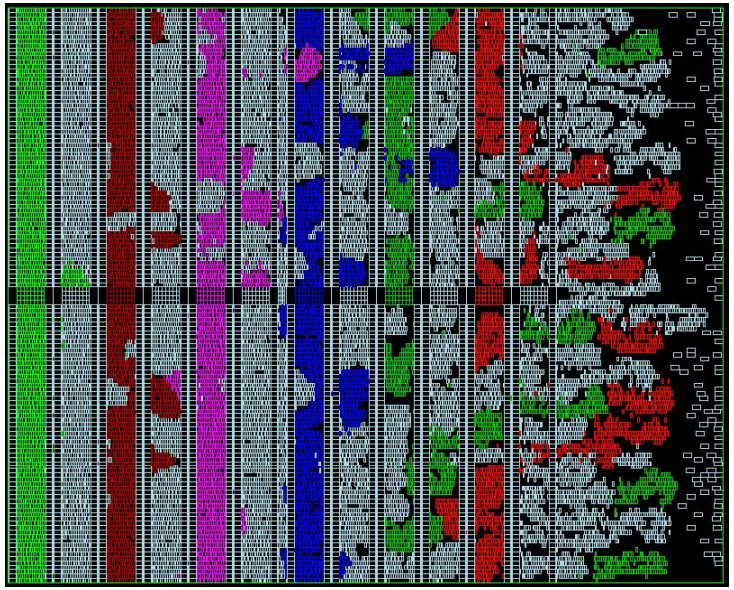

## Physical Design



# Design 1 Placement Cont.

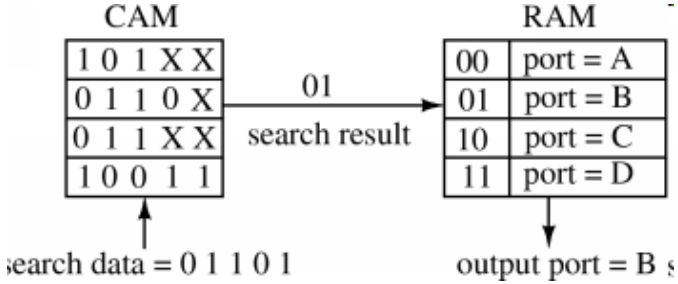
- **Highlighted rows:**
  - Helps track changes between manual solution and placed solution
  - Highlights areas of suboptimality
- **Observations**
  - Clustering impact high in areas of more available whitespace
    - Ex: Red stack highly segmented
  - Legalization an issue in areas of little whitespace
    - Ex: Left green stack



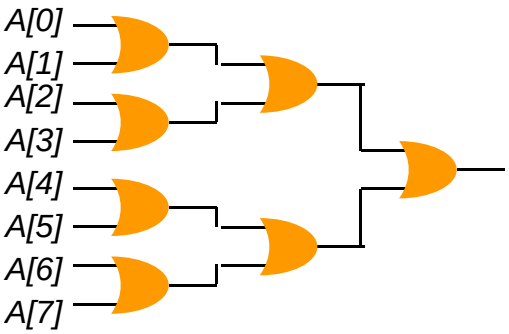


# Design 2: Structured Trees such as AND/OR Logic

- **Logic 101…**
  - Load/Store Queue (simple memory)
  - Content Addressable Memories
  - Greater than/Less than
  - Basic ALU Operations
- **Common structure repeated regularly**
- **Standard cells can be interchanged to match any of these functions**

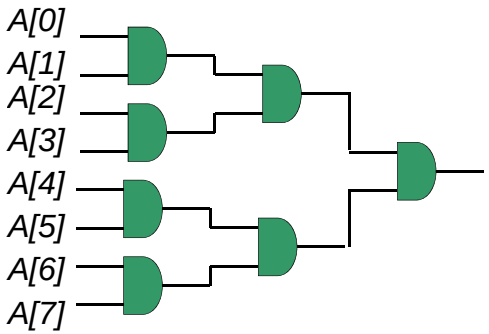



*Any 1's?*



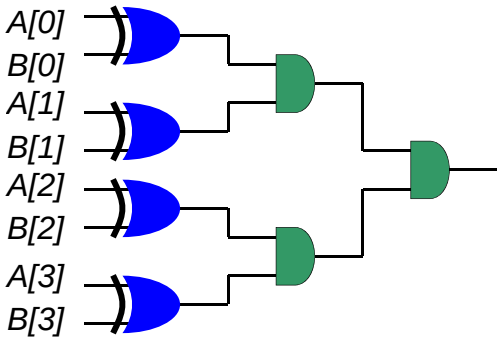

*OR_REDUCE(A[0:7])*

*All 1's?*




*AND_REDUCE(A[0:7])*

*Does A equal B?*

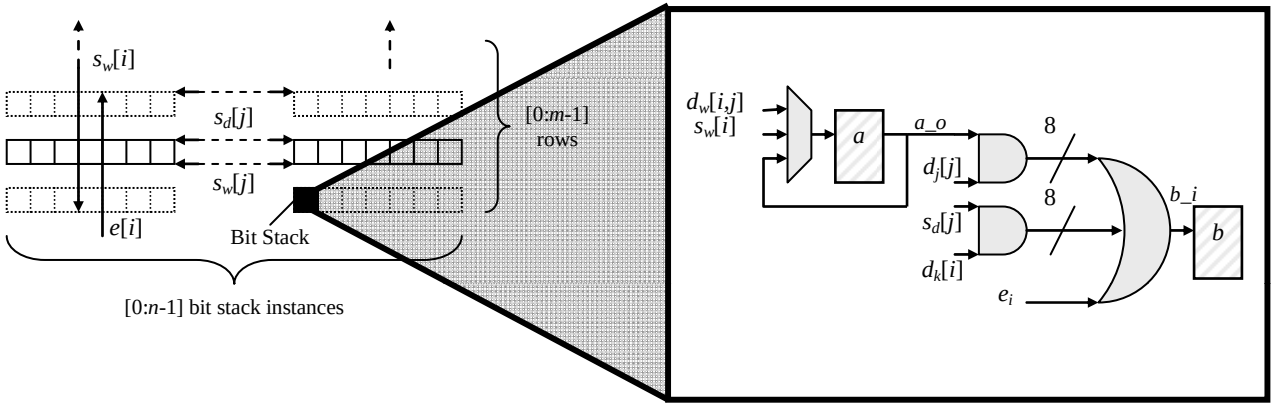



*A[0:3] == B[0:3]*

# Design 2: Placement

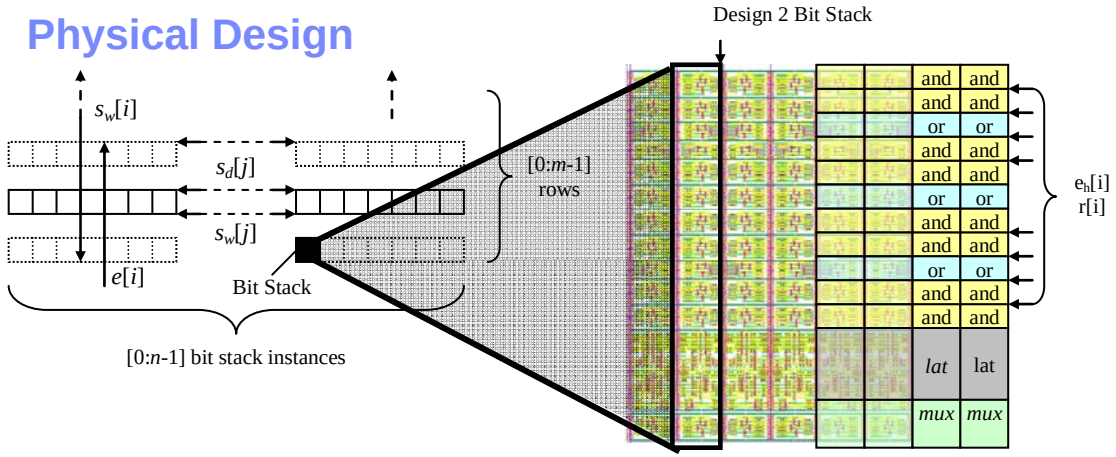
- **Replicable with**
  - Large OR-trees
  - Large AND-trees
  - Compare Logic

Logical Design



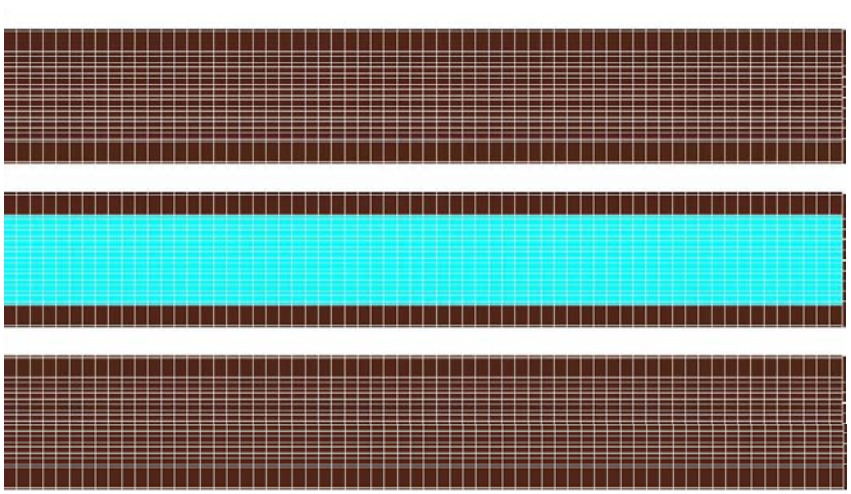

- **Characteristics**
  - Many global connections between each bit stack
  - Few local connections between each bit stack

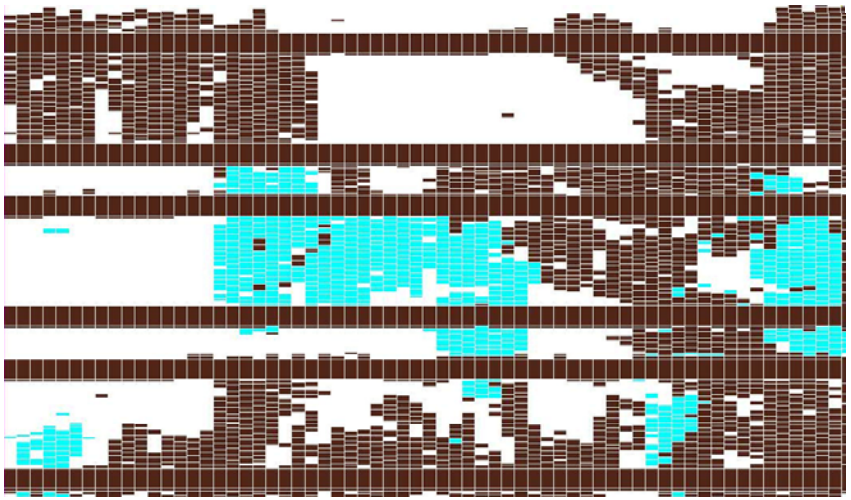
Physical Design



# Design 2 Placement Continued

- **Custom design verses automatic placement**
- **Results:**
  - Clustering causes logic to clump together
  - Timing and Congestion increase in the process of being quantified
  - Placement unaware of logical partitions
  - Currently: Fixed Latches Improve Overall Placement Results





# Errata to Published Benchmarks

- **Goal: generate an end to end benchmark flow solving the dataflow design problem**
- **Changes**
  - Added Clocking Signals
    - reduces HPWL Ratio because latches are fixed
    - Needed for future timing work
  - Pin Locations and Pin Count:
    - Improved Placement
    - Improved Pin Count, more pins for a flatter netlist
  - Simplification – Removed some control logic to focus on datapath placement
- **Why?**
  - Ongoing Research Quantifying Other Areas of Suboptimality
    - Routing
    - Power
    - Delay
- **Published at:** http://vlsicad.eecs.umich.edu/BK/spb
  - Special thanks to Professor Igor Markov

# Results: Base Case

- Design 1
  - Most placers generate overlaps
  - ntuPlace3 Aborted
  - CAPO: best overall HPWL
- Design 2
  - Most placers generate overlaps
  - Overall better than design 1
  - ntuPlace3: best overall HPWL

| | Design 1 | | | Design 2 | | |
|---|---|---|---|---|---|---|
| Placer | TWL | TWL Ratio | Run Time (s) | TWL | TWL Ratio | Run Time (s) |
| Custom | 11000365 | 1.00 | n/a | 8642097 | 1.00 | n/a |
| Capo | 15945589* | 1.45* | 1453.9 | 14381067* | 1.66* | 1430.6 |
| mPL6 | 18290965* | 1.66* | n/a | n/a | n/a | n/a |
| ntuPlace3 | n/a | n/a | n/a | 11110251* | 1.29* | 533.0 |
| APlace* | n/a | n/a | n/a | n/a | n/a | n/a |
| Dragon | 52926316 | 4.81* | 2350.18 | 34711167 | 4.02* | 2692.0 |
| FastPlace | 16336840* | 1.49* | 194.9 | n/a | n/a | n/a |

- ** Completed with Overlaps*
- *ntuPlace3: Aborts during global placement for Design 1*
- *n/a: Did not complete*

# Results: Whitespace

## Design 1 Test Cases

- * generated overlaps
- ntuPlace3 Aborted
- CAPO: best overall HPWL

| Whitespace | 92.5 | 89.0 | 85.8 | 82.8 | 80.1 | 77.4 | 74.0 | 71.9 |
|---|---|---|---|---|---|---|---|---|
| Capo | 1.45* | 1.49* | 1.24* | 1.28* | 1.14* | 1.18* | 1.12* | 1.11* |
| mPL6 | 1.66* | 1.65* | 1.64* | 1.66* | 1.64* | 1.63* | 1.76* | 1.73* |
| ntuPlace3 | - | - | - | - | - | - | - | - |
| aPlace | - | - | - | - | - | - | - | - |
| Dragon | 4.81 | 5.00 | 5.39 | 5.88 | 5.83 | 5.91 | 6.56 | 7.37 |
| FastPlace | 1.49* | 1.33* | 1.31* | 1.30* | 1.27* | 1.27* | 1.29* | 1.30* |

## Design 2 Test Cases

- Best HPWL seen at 10% to 15% whitespace
- ntuPlace3: best overall HPWL

| Whitespace | 95.5 | 93.6 | 89.5 | 85.3 | 81.5 | 78.1 | 75.2 | 72.2 |
|---|---|---|---|---|---|---|---|---|
| Capo | 1.66* | 1.24* | 1.17* | 1.18* | 1.18* | 1.20* | 1.20* | 1.21* |
| mPL6 | - | 1.19* | 1.15* | 1.72* | 1.15* | 1.16* | 1.17* | 1.18* |
| ntuPlace3 | 1.29* | 1.12 | 1.14 | 1.13 | 1.20 | 1.15 | 1.16 | 1.24 |
| aPlace | - | - | - | - | - | - | - | - |
| Dragon | 4.02 | 4.24 | 4.49 | 4.81 | 5.09 | 5.33 | 5.60 | 5.93 |
| FastPlace | - | 1.26 | 1.15 | 1.15 | 1.17 | 1.19 | 1.20 | 1.21 |

# Future Work

- **High Density Legalization**
  - High utilization of datapath design difficult to solve efficiently
- **Routing Aware Placement**
  - Easy to pack, hard to route
- **Structural Analysis**
  - Understanding logic structure can improve results
  - How do we quickly evaluate the logical structure?
- **Datapath Extraction**
  - Datapath Components in Traditional Random Logic
  - Simultaneous Optimization of Both Styles