VLSI architecture, synthesis & technology

UCLA

Scheduling and Physical Design

Jason Cong

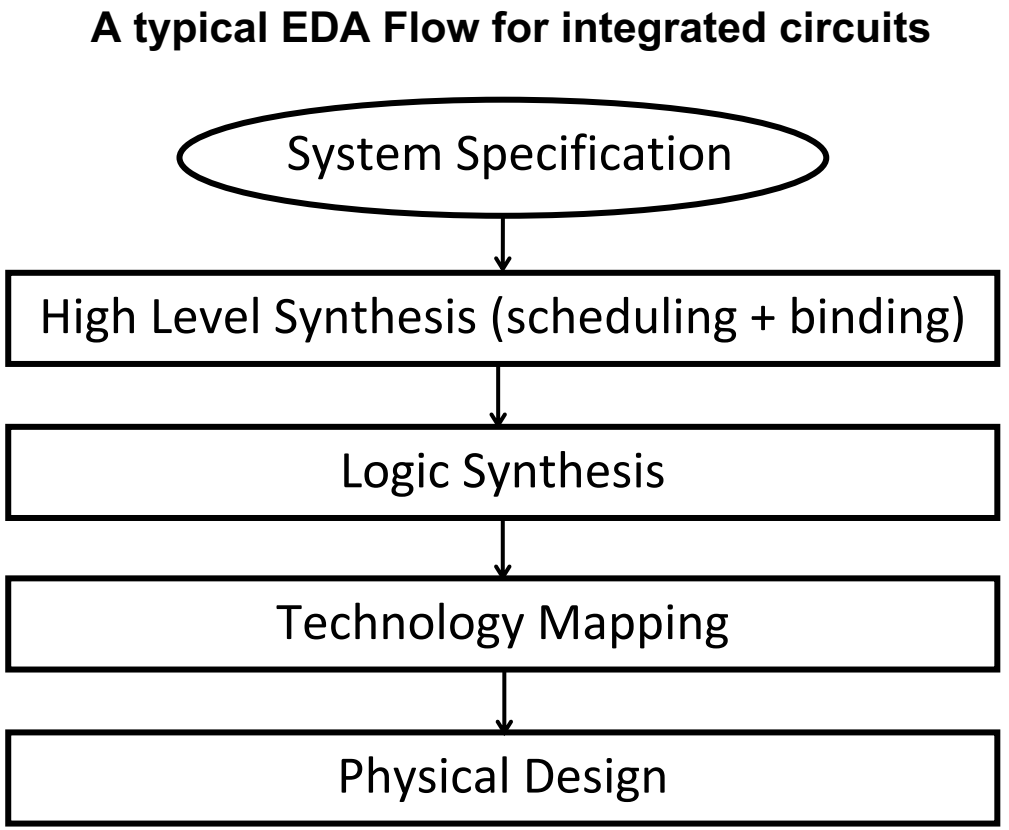
Volgenau Chair for Engineering Excellence, UCLA Computer Science

Director, Center for Domain-Specific Computing (CDSC)

https://vast.cs.ucla.edu/people/faculty/jason-cong

Why This Title/Topic?

- **Scheduling:**
  - **Determine when each operation takes place**
  - **Part of high-level synthesis -- the first stage of the design flow**
- **Physical design:**
  - **Determine where each operation takes place and the best way to interconnect them**
  - **The last stage of the design flow**



It Goes Back to My First Paper with Martin [ICCAD'1987]

A New Approach to Three- or Four-Layer Channel Routing

JINGSHENG CONG, D. F. WONG, AND C. L. LIU, FELLOW, IEEE

Abstract—We present in this paper a new approach to the three- or four-layer channel routing problem. Since two-layer channel routing has been well studied, there are several two-layer routers which can produce optimal or near optimal solutions for almost all the practical problems. We develop a general technique which transforms a two-layer routing solution systematically into a three-layer routing solution. This solution transformation approach is different from previous approaches for three-layer and multilayer channel routing. Our router performs well in comparison with other three-layer channel routers proposed thus far. In particular, it provides a ten-track optimal solution for the famous Deutsch's difficult example, whereas other well known three-layer channel routers required 11 or more tracks. We extend our approach to four-layer channel routing. Given any two-layer channel routing solution without an unrestricted dogleg that uses w tracks, our router can provably obtain a four-layer routing solution using no more than $\lceil w/2 \rceil$ tracks. We also give a new theoretical upper bound $\lceil d/2 \rceil + 2$ for arbitrary four-layer channel routing problems.

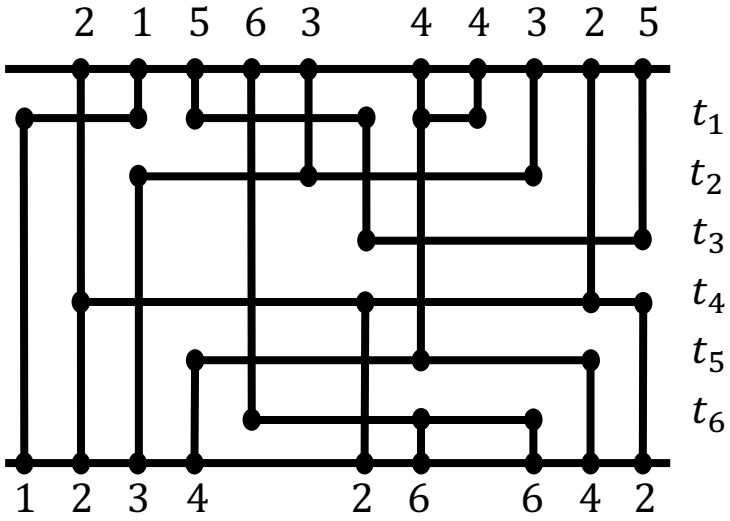
I. Introduction

A KEY PROBLEM in VLSI layout design and implementation is the channel routing problem. The two-layer channel routing problem has been studied extensively in the literature. Recently, a 1-megabit DRAM designed by Taguchi *et al.* uses four routing layers, three layers of polysilicon and one layer of metal. Thus, the design and implementation of channel routing algorithms using a small number of layers (usually three or four layers) are not only practical, but also are becoming more and more important.

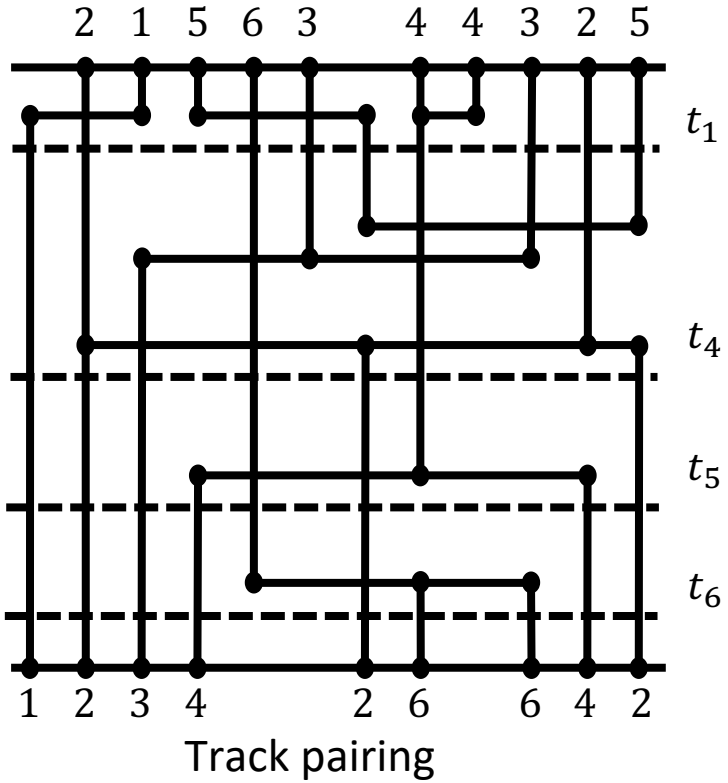
The multilayer channel routing problem has been studied in the literature. Chen and Liu [5] presented a three-layer channel router based on the net merging method used by Yoshimura and Kuh [22] for two-layer channel routing. Bruell and Sun [3] designed a ''greedy'' router for three-layer channel routing and obtained the first 11-track solution for Deutsch's difficult example. Braun *et al.* [2] implemented a multilayer channel router which divides layers into several groups. Each group contains two or three layers and routing for each group is done by the extended two-layer router YACR2 [20]. Enbody and Du [11] developed a multilayer router using leading column heuristics and limited backtracking. As for theoretical results, Hambrusch [15] obtained some near-optimal upper bounds for the case of two terminal nets allowing mixed wiring on the same layer. Brady and Brown [1] proposed

Three-Layer Channel Routing from Two-Layer Solutions

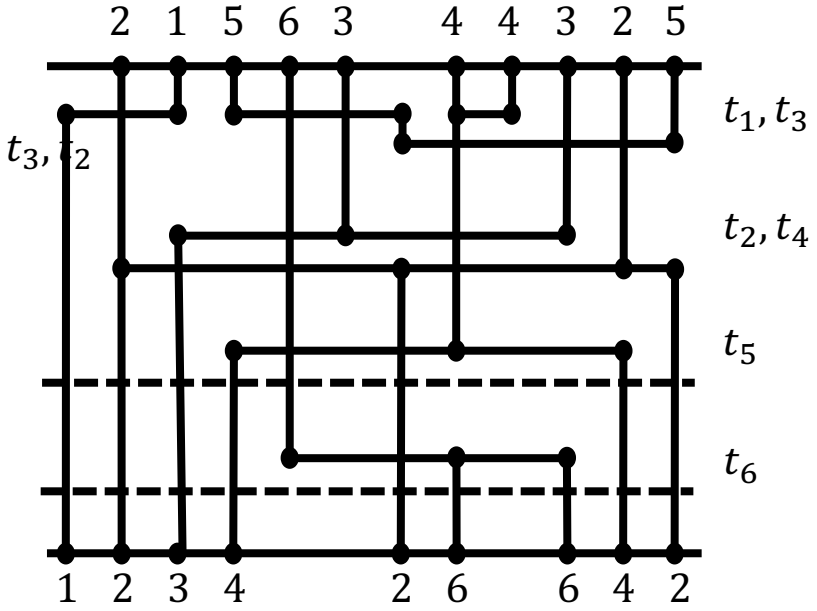
- **Goal: Minimize the number of tracks in a 3L solution**
- **Model: HVH model – the 1st and 3rd layers route horizontal wires and the 2nd layer routes vertical wires**



Two-layer channel routing

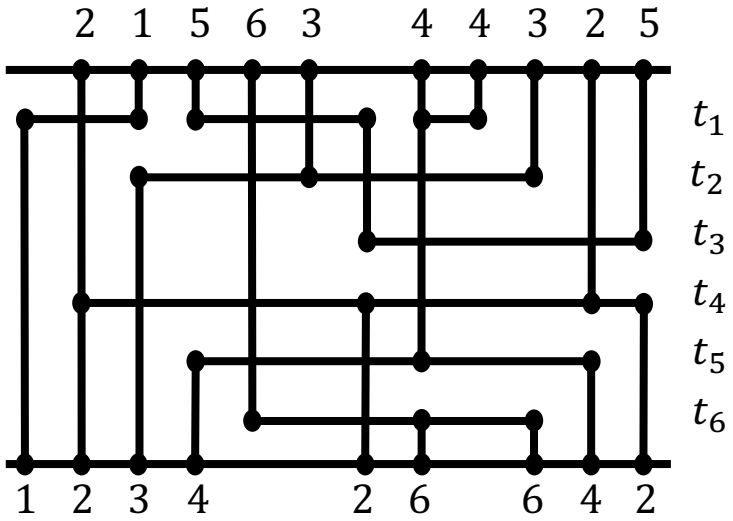


Track pairing



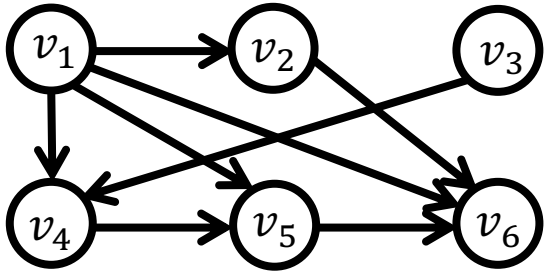
Perfect track pairing

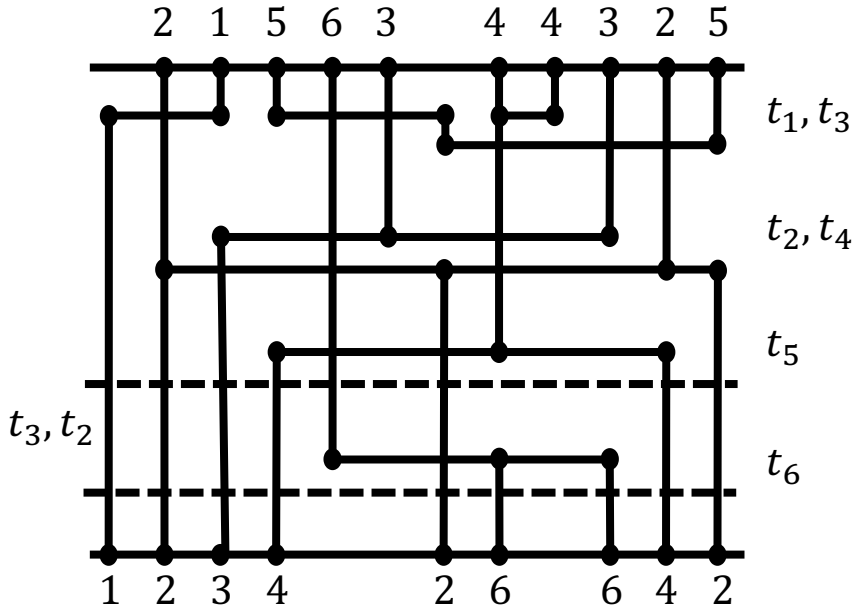
How to Compute an Optimal Track Ordering?



Two-layer channel routing

Track ordering graph (ToG) for 3L channel routing



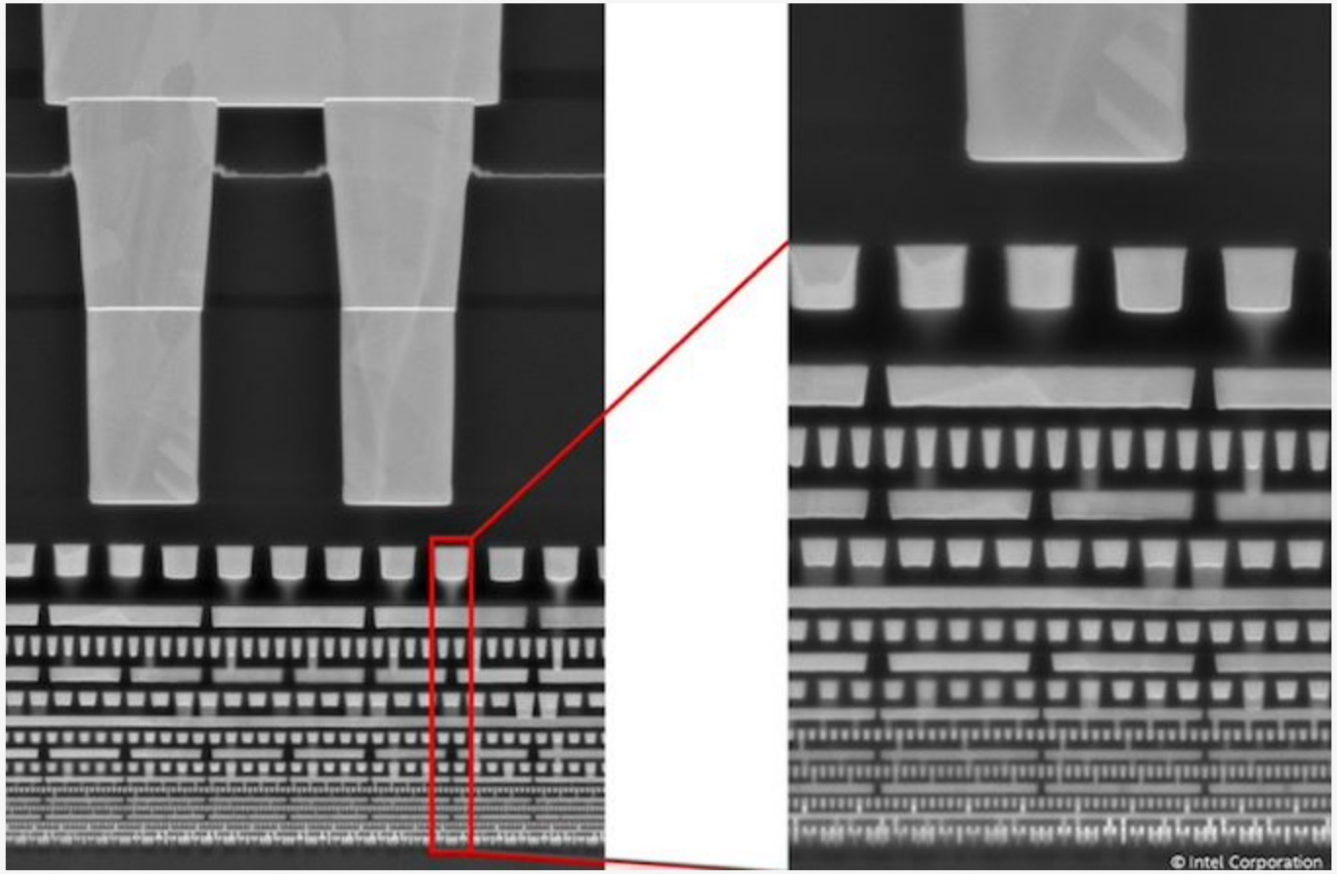


Perfect track pairing

- **Theorem: An optimal track ordering ⇔ an optimal two-processor *scheduling* solution with ToG as the task dependency graph [ICCAD'87]**
- **Together with a few other optimization, we were the first to obtain an optimal solution to the Deutsch's Difficult Example**
- **Three-layer metal technology was indeed introduced two years later in Intel 486 (with over 1M transistors)**

Fast Forward 35 Years: IC Technology Today

Example: Intel 4 technology with 17 metal layers to be fabricated in EUV



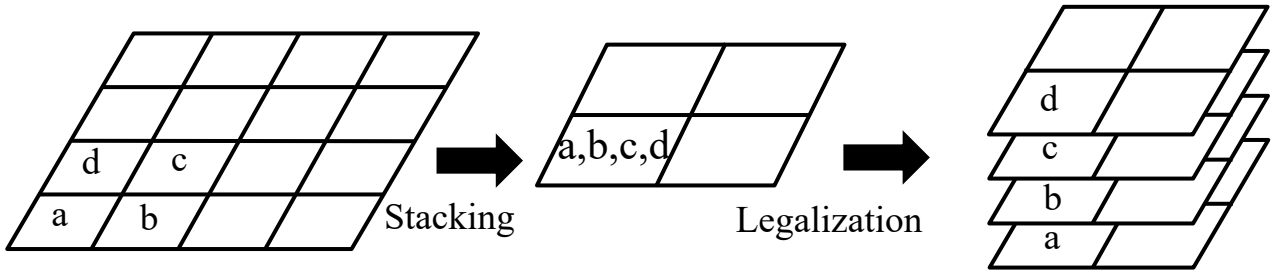
Intel 4 Metal Stack		
Layer	**Metal**	**Pitch**
Fin	-	30 nm
Gate	Tungsten	50 nm
Metal 0	Copper w/Cobalt Cladding	30 nm
Metal 1	Copper w/Cobalt Cladding	50 nm
Metal 2	Copper w/Cobalt Cladding	45 nm
Metal 3	Copper w/Cobalt Cladding	50 nm
Metal 4	Copper w/Cobalt Cladding	45 nm
Metal 5, 6	Copper	60 nm
Metal 7, 8	Copper	84 nm
Metal 9, 10	Copper	98 nm
Metal 11, 12	Copper	130 nm
Metal 13, 14	Copper	160 nm
Metal 15	Copper	280 nm
Giant Metal 0	Copper	1080 nm
Giant Metal 1	Copper	4000 nm

https://www.anandtech.com/show/17448/intel-4-process-node-in-detail-2x-density-scaling-20-improved-performance

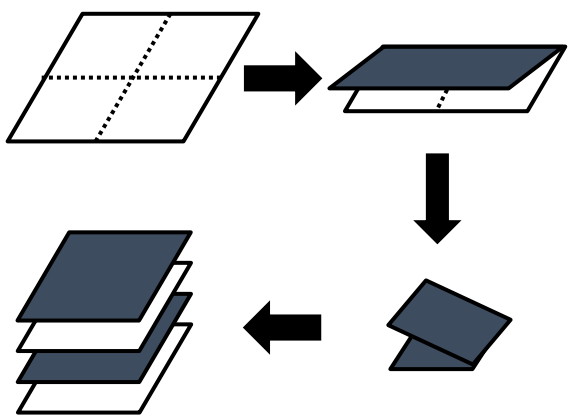
A New Life of the Transformation-Based Approach

- Although the transformation-based approach was not used in many-layer routing
- It was used in 3D-IC placement 20 years later

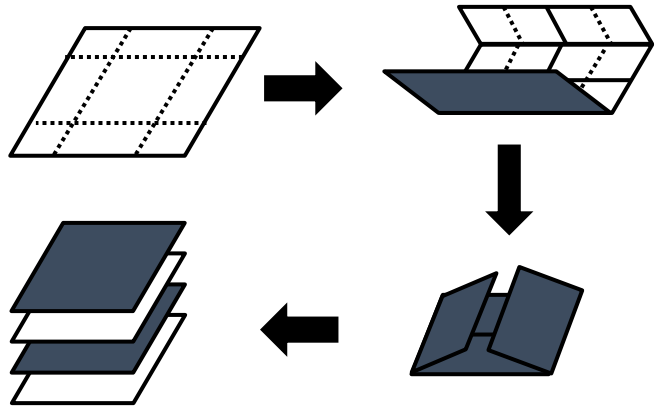
Transformation-Based Approach for 3D-IC Placement



Local stacking transformation



Folding-2 transformation



Folding-4 transformation

J. Cong, G. Luo, J. Wei, and Y. Zhang. Thermal-Aware 3D IC Placement via Transformation. ASP-DAC 2007, Yokohama, Japan, **(The ASP-DAC 2017 Ten-Year Retrospective Most Influential Paper Award).**

The Most Significant Collaboration with Martin

- The Best man at Martin’s wedding

Decade of 1990s: Working Side-by-Side with Martin on Interconnect Optimization

Optimal Wiresizing Under the Distributed Elmore Delay Model

Jason Cong and Kwok-Shing Leung
Department of Computer Science
University of California, Los Angeles, CA 90024

Abstract

In this paper, we study the optimal wiresizing problem under the distributed Elmore delay model. We show that the optimal wiresizing solutions satisfy a number of interesting properties, including the separability, the monotone property, and the dominance property. Based on these properties, we develop a polynomial-time optimal wiresizing algorithm for arbitrary interconnect structures under the distributed Elmore delay model. Extensive experimental results show that our wiresizing solution reduces interconnection delay by up to 51% when compared to the uniform-width solution of the same routing topology. Furthermore, compared to the wiresizing solution based on a simpler RC delay model in [7], our wiresizing solution reduces the total wiring area by up to 28% while further reducing the interconnection delays to the timing-critical sinks by up to 12%.

1 Introduction

As the VLSI fabrication technology reaches submicron device dimension and gigahertz frequency, interconnection delay has become the dominant factor in determining circuit speed [9, 14]. The analysis in [15] and [7] showed that in the conventional VLSI technology, interconnection delay is determined by the product of the driver resistance and the total wire capacitance. As a result, the minimum interconnection delay is achieved when the routing tree is an optimal Steiner tree with the minimum wire width for each segment (since it has the minimum total wire capacitance). Therefore, conventional global and detailed routers aimed at generating minimum-width Steiner routing trees using the least total wirelength. However, as we reduce the device dimension, the driver resistance becomes smaller and the wire resistance becomes larger, which results in a much larger resistance ratio (defined to be the ratio of the driver resistance versus the unit wire resistance). In this case, the *distributed* nature of the interconnect structure must

Experimental results showed that the algorithm in [7] can construct A-trees which are at most 4% within the optimal, and achieve interconnection delay reduction by as much as 66% when compared to the best-known Steiner routing topology. When the critical-path information is available, the critical sink routing approaches in [2] reduce the delays to specified sinks substantially.

Although steady progress has been made in optimizing interconnect topology design for delay minimization, there were very few works on wiresizing optimization for high-performance interconnect designs. Wiresizing was used by Fisher and Kung [11] in H-tree clock routing. Recently, Cong, Leung, and Zhou [7] developed an optimal wiresizing algorithm based on minimizing an upper bound of the delay in a distributed RC tree proposed by Rubinstein, Penfield and Horowitz [13], which is given by:

$$t = \sum_{\text{all nodes } k} R_k \cdot c_k \qquad (1)$$

where R_k is the path resistance between the source and the node k and c_k is capacitance at the node k. This upper-bound delay model was chosen in [7] because it simplifies the wiresizing optimization. However, the simplicity of this delay model also results in several drawbacks. First, it provides only an upper bound of the worst-case RC delay in the routing tree and does not distinguish the delays at different sinks. Therefore, it is impossible to optimize the wiresizing solution to reduce the delays to the specific timing-critical sinks. Moreover, since this model tends to over-estimate the delays at many sinks in the routing tree, it often results in unnecessary over-sizing of many wire segments. Oversized wires not only occupy more routing spaces, but also increase the mutual capacitance and inductance between different signal nets. Thus, there is a strong need to develop optimal wiresizing algorithms under more accurate interconnection delay models

ICCAD 1993

Optimal Wire-Sizing Formula Under the Elmore Delay Model *

Chung-Ping Chen, Yao-Ping Chen, and D. F. Wong
Department of Computer Sciences, University of Texas, Austin, Texas 78712

Abstract

In this paper, we consider non-uniform wire-sizing. Given a wire segment of length $\mathcal{L}$, let $f(x)$ be the width of the wire at position x, $0 \le x \le \mathcal{L}$. We show that the optimal wire-sizing function that minimizes the Elmore delay through the wire is $f(x) = ae^{-bx}$, where $a > 0$ and $b > 0$ are constants that can be computed in $O(1)$ time. In the case where lower bound ($L > 0$) and upper bound ($U > 0$) on the wire widths are given, we show that the optimal wire-sizing function $f(x)$ is a truncated version of ae^{-bx} that can also be determined in $O(1)$ time. Our wire-sizing formula can be iteratively applied to optimally size the wire segments in a routing tree.

1 Introduction

As VLSI technology continues to scale down, interconnect delay has become the dominant factor in deep submicron designs. As a result, wire-sizing plays an important role in achieving desirable circuit performance. Recently, many wire-sizing algorithms have been reported in the literature [1, 2, 4, 5, 7]. All these algorithms size each wire segment uniformly, i.e., identical width at every position on the wire. In order to achieve non-uniform wire-sizing, existing algorithms have to chop wire segments into large number of small segments. Consequently, the number of variables in the optimization problem is increased substantially and thus results in long runtime and large storage.

In this paper, we consider non-uniform wire-sizing. Given a wire segment W of length $\mathcal{L}$, a source with driver resistance R_d, and a sink with load capacitance C_L. For each $x \in [0, L]$, let $f(x)$ be the wire width of W at position x. Figure 1 shows an example. Let r_0 and c_0 be the respective wire resistance and wire capacitance per unit square. Let D be the Elmore delay from the source to the sink of W. We show that the optimal wire-sizing function $f(x)$ that minimizes D satisfies an ordinary differential equation which can be analytically solved. We have $f(x) = ae^{-bx}$, where $a > 0$ and $b > 0$ are constants that can be computed in $O(1)$ time. These

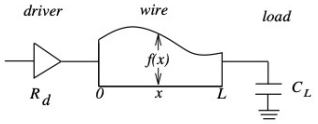


Figure 1: Non-uniform wire-sizing.

$= \frac{\mathcal{L}}{n}$. Let x_i be $i\triangle x$, $1 \le i \le n$. The capacitance and resistance of wire segment i can be approximated by $c_0 \triangle x f(x_i)$ and $r_0 \triangle x / f(x_i)$, respectively. Thus the Elmore delay through W can be approximated by

$$D_n = R_d(C_L + \sum_{i=1}^{n} c_0 f(x_i)\triangle x) + \sum_{i=1}^{n} \frac{r_0 \triangle x}{f(x_i)} (\sum_{j=i}^{n} c_0 f(x_j)\triangle x + C_L).$$

The first term is the delay in the driver, which is given by the driver resistance R_d multiplied by the total capacitance of W and C_L. The second term is the sum of the delay in each wire segment i, which is given by its own resistance $r_0 \triangle x / f(x_i)$ multiplied by its downstream capacitance $\sum_{j=i}^{n} c_0 f(x_j)\triangle x + C_L$. As $n \to \infty$, $D_n \to D$ where

$$D = R_d(C_L + \int_0^{\mathcal{L}} c_0 f(x)dx) + \int_0^{\mathcal{L}} \frac{r_0}{f(x)} \, (\int_x^{\mathcal{L}} c_0 f(t)\, dt + C_L)\, dx$$

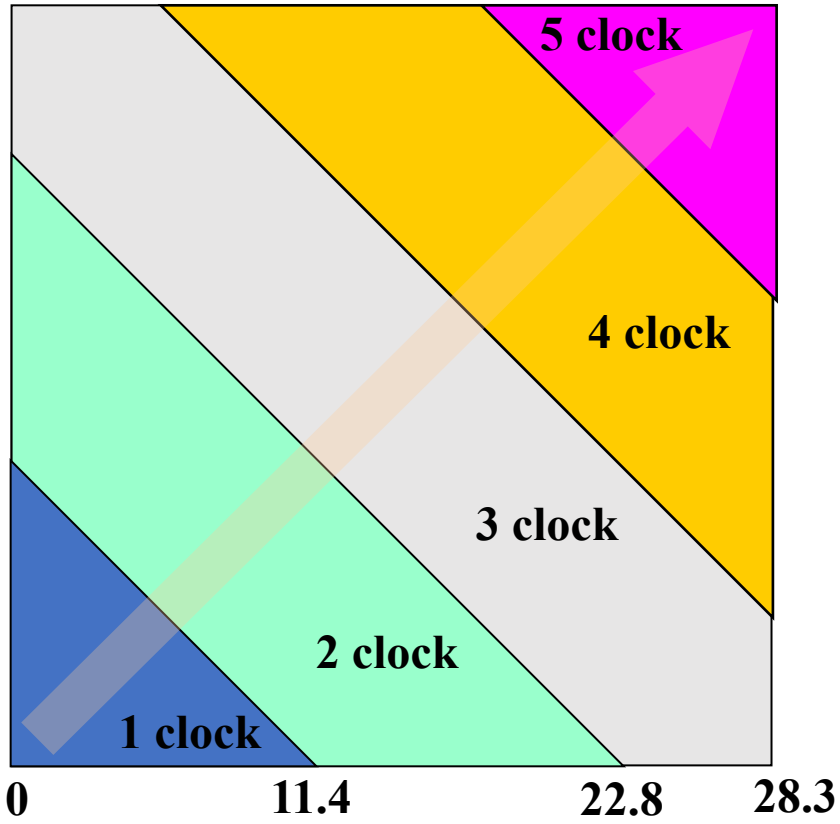
is the Elmore delay through W.

In this section, we derive closed-form formula for the optimal wire-sizing function $f(x)$. We consider two cases: *unconstrained* and *constrained wire-sizing*. In unconstrained wire-sizing, there is no bound on the value of $f(x)$; i.e. we determine $f : [0, \mathcal{L}] \to (0, \infty)$ that minimizes D. In constrained wire-sizing, we are given $L > 0$ and $U < \infty$, and require that $L \le f(x) \le U$, $0 \le x \le \mathcal{L}$; i.e., we determine $f : [0, \mathcal{L}] \to [L, U]$ that minimizes D.

DAC 1996

Decade of 2000s: Multi-Cycle On-Chip Communication and High-Level Synthesis

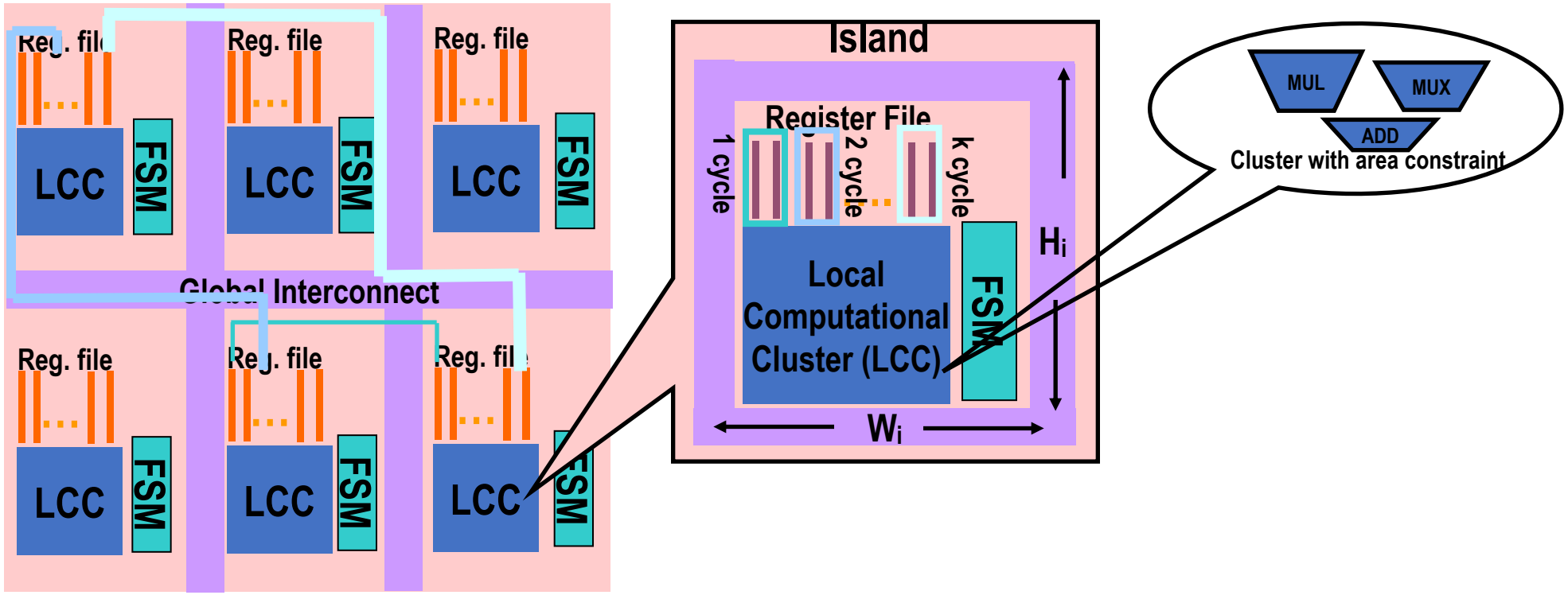
- Interconnect delays dominate the timing in deep submicron technology
- Single-cycle full chip synchronization is no longer possible



- ITRS'01 70nm Tech
- 5.63 G Hz across-chip clock
- 800 mm^2 (28.3mm x 28.3mm)
- IPEM BIWS estimations
  - Buffer size: 100x
  - Driver/receiver size: 100x
- From corner to corner:
  - at semi-global layer (Tier 3)
  - can travel up to 11.4mm in one cycle
  - need 5 clock cycles

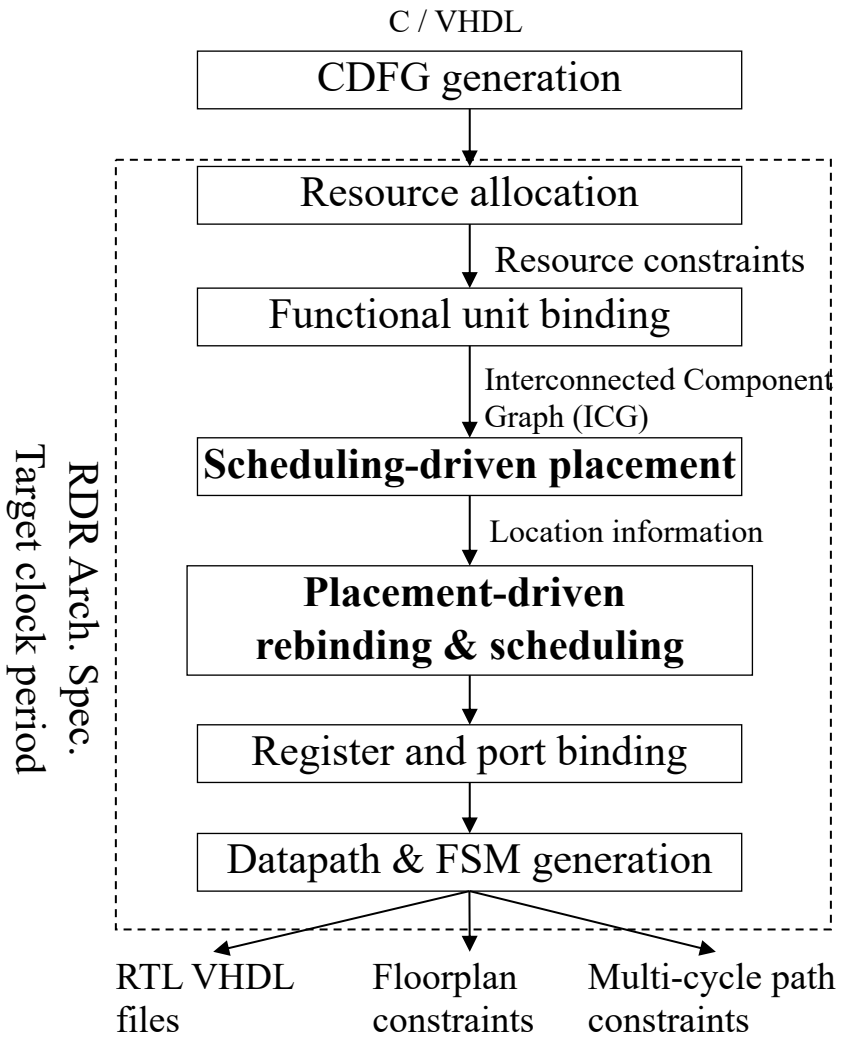
J. Cong, et al, T-CAD 2004

Our Proposal: Regular Distributed Register (RDR) Architecture [T-CAD'2004]



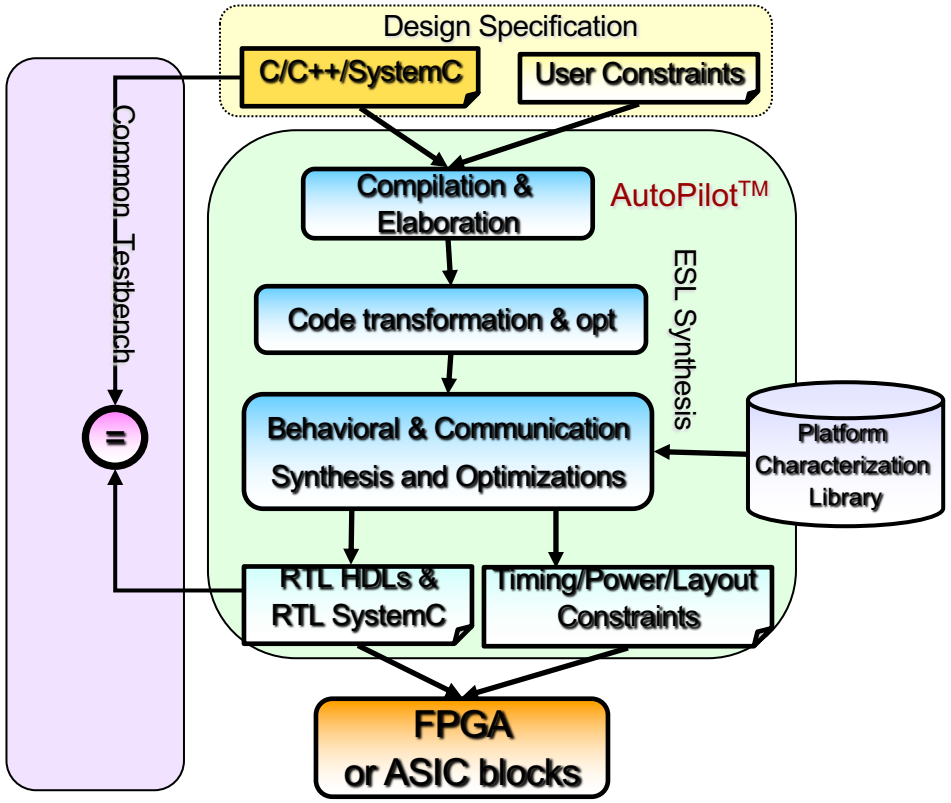
- Use register banks:
  - Registers in each island are partitioned to k banks for 1 cycle, 2 cycle, … k cycle interconnect communication in each island
- Highly regular
- Goal: high-frequency designs

MCAS: Placement-Driven Architectural Synthesis Using RDR Architecture



- **Multi-Cycle Communication Architectural Synthesis (MCAS) System**
  - **Scheduling-driven placement**
  - **Placement-driven rescheduling & rebinding**
- **Limitations**
  - **The high-level synthesis (HLS) engine was not robust**
  - **Did not consider interconnect pipelining**
  - **Regularity imposes some overhead**
- **This idea has to wait for another 17 years to mature**

Development of Robust and Scalable HLS Tool



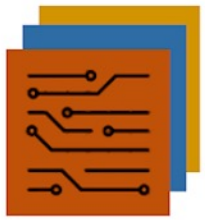
- xPilot (UCLA 2006) -> AutoPilot (AutoESL) -> Vivado HLS (Xilinx 2011-)
  - LLVM based compilation
  - Platform-based C to RTL synthesis
  - Synthesize pure ANSI-C and C++, GCC-compatible compilation flow leveraging LLVM framework
  - Full support of IEEE-754 floating point data types & operations
  - Efficiently handle bit-accurate fixed-point arithmetic
  - SDC-based scheduling
  - Automatic memory partitioning
- …

QoR matches or exceeds manual RTL for many designs

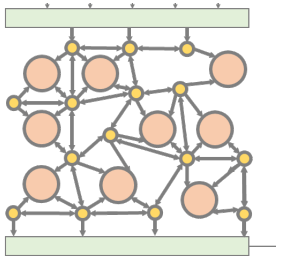
TCAD April 2011 (keynote paper) “High-Level Synthesis for FPGAs: From Prototyping to Deployment”

HLS with Automated Interconnect Pipelining

- User specify how the HLS modules or IP blocks are connected in dataflow
- Co-optimizes interconnect pipelining and design partitioning => 2X Fmax
- Parallel placement & routing => 5-7X productivity

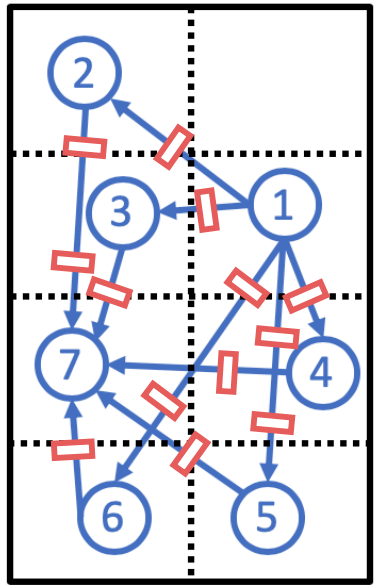


IPs
(Verilog, HLS, IP, …)



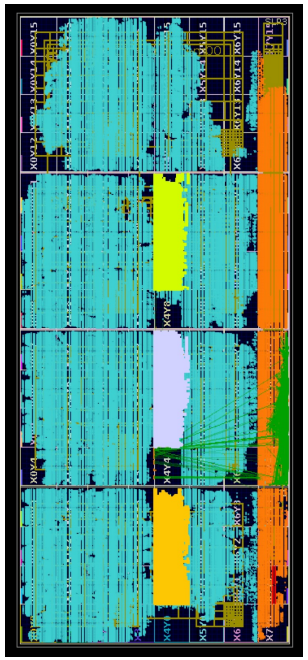
Logical Interconnects
(LiteX, Vivado IPI, etc.)

Interconnect Synthesis & Physical Optimization (FPGA'21 Best Paper, 2X Frequency)



Layout-Guided
Re-synthesis & Composition

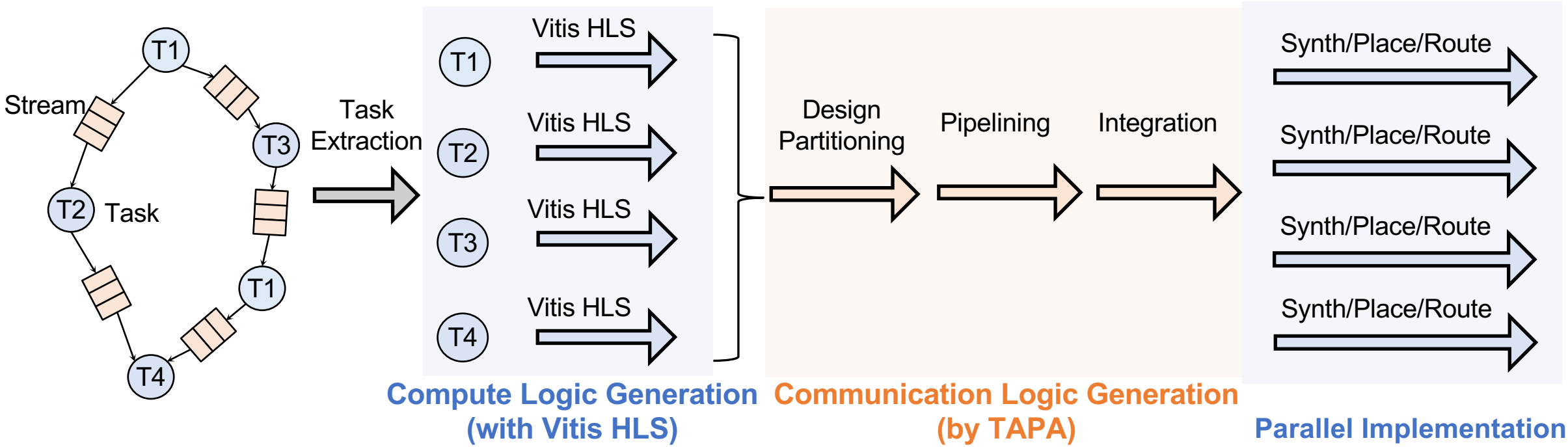
Parallel Placement & Routing (FPGA'22 Best Paper, 5-7X Compilation Speed Up)



Final Bitstream

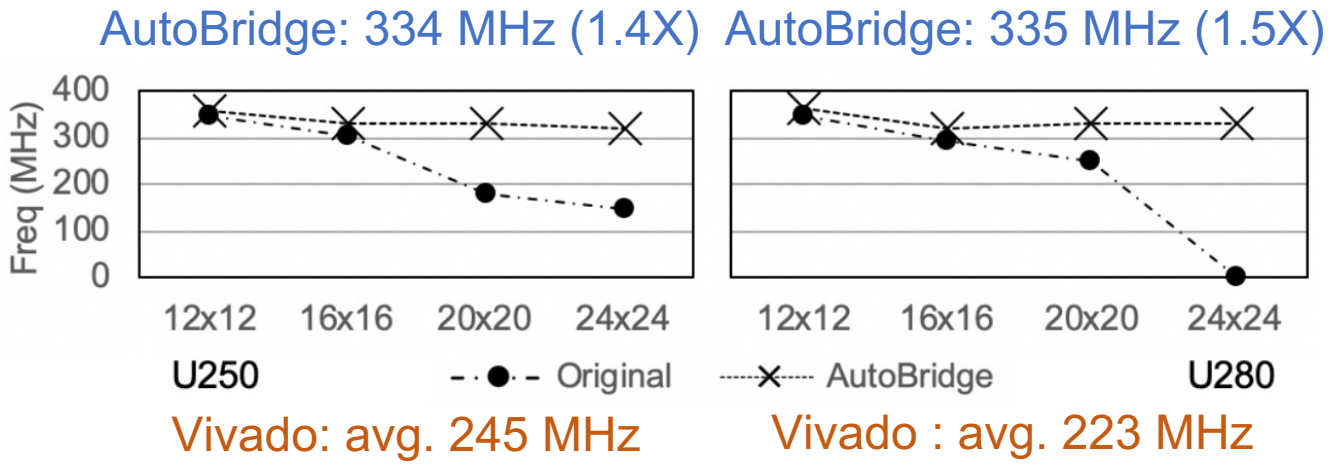
Dataflow Design using TAPA [FCCM'21, TRETS'23]

- TAPA is a task-Parallel HLS framework
- Integrated AutoBridge and RapidStream
- Extends Vitis HLS with additional APIs

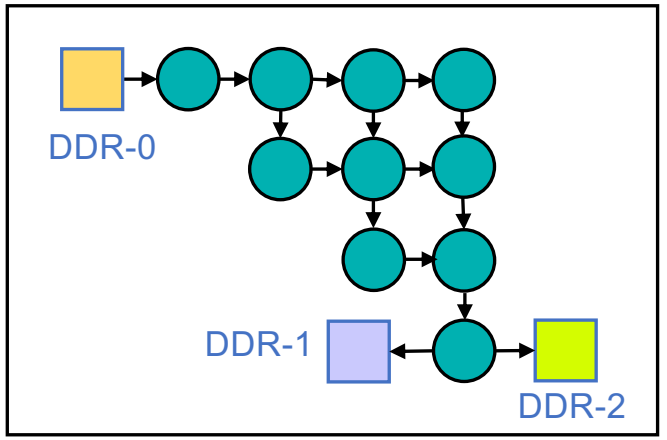


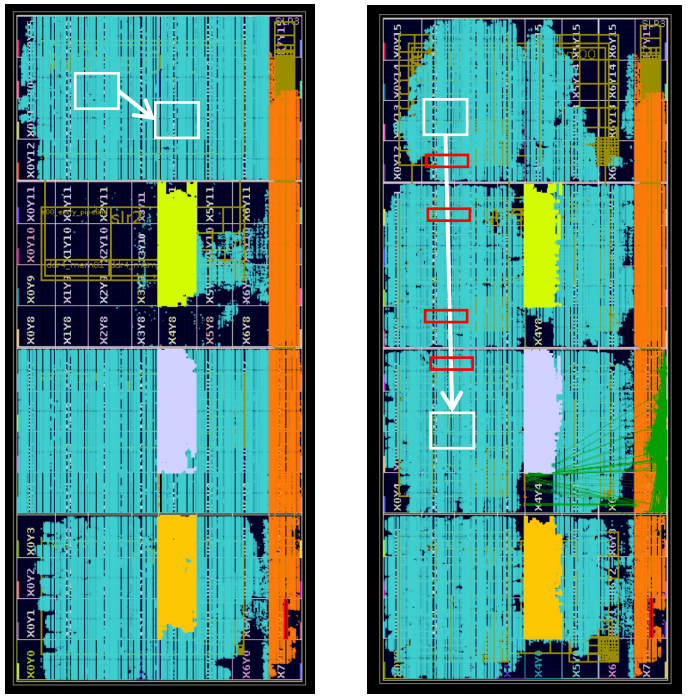
Case Study

- Systolic array Gaussian elimination, 8 configurations



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

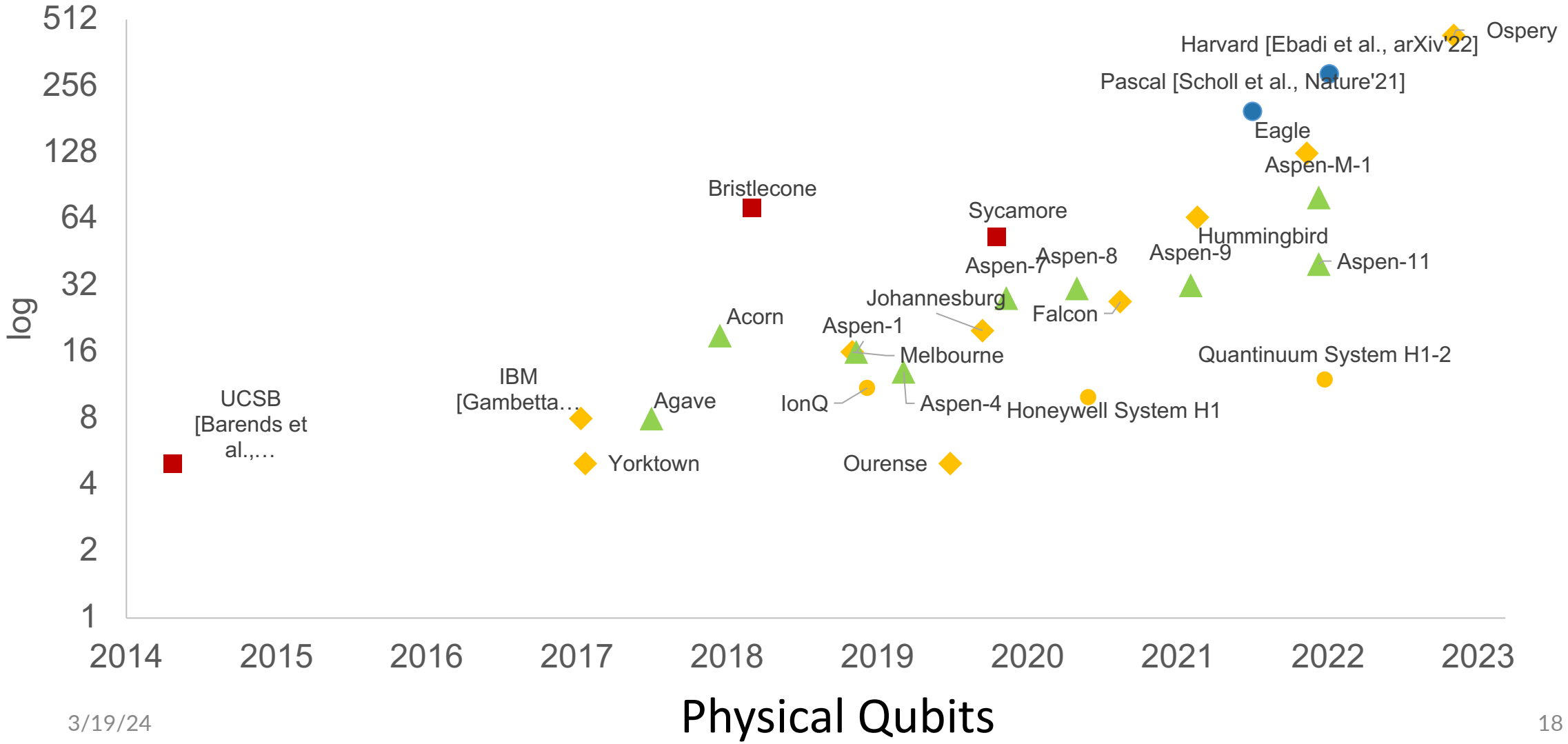




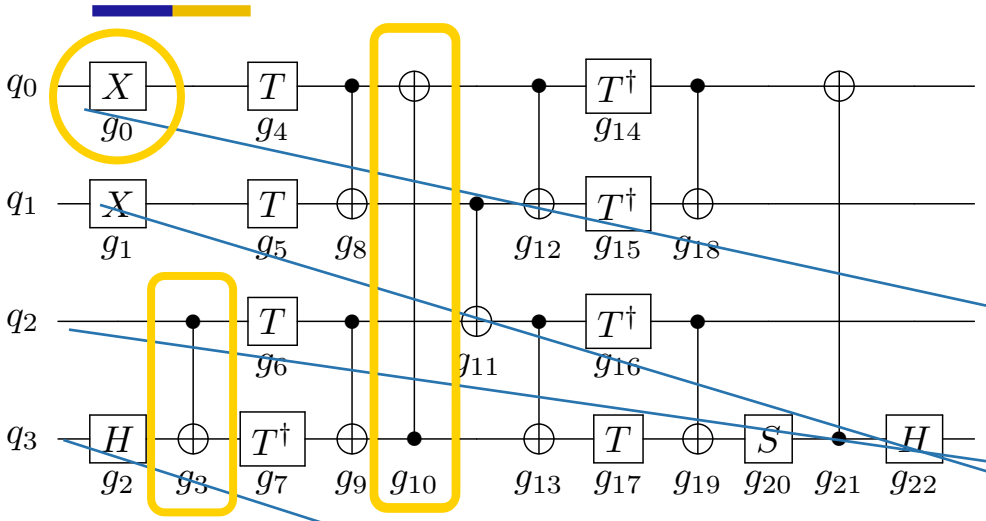
Vivado AutoBridge

Comparison of the 24x24 Design on U250

Decade of 2020s: Compilation for Quantum Computing



Quantum Layout Synthesis (QLS)



Input Circuit of Adder

```
# Input quantum program
x q[0];
x q[1];
h q[3];
cx q[2], q[3];
t q[0];
```

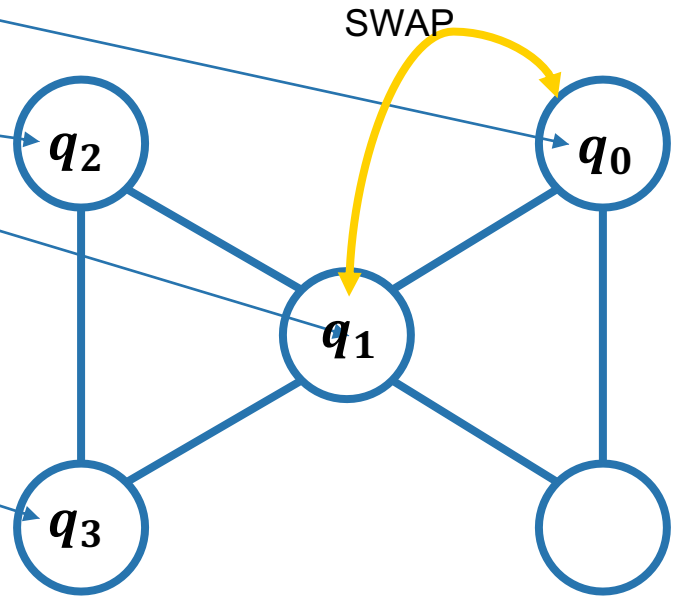
…

† means Hermitian conjugate, which is straightforward once we have the original gate implementation.

CX on a pair of adjacent qubits, OK.

CX on a pair of non-adjacent qubits!

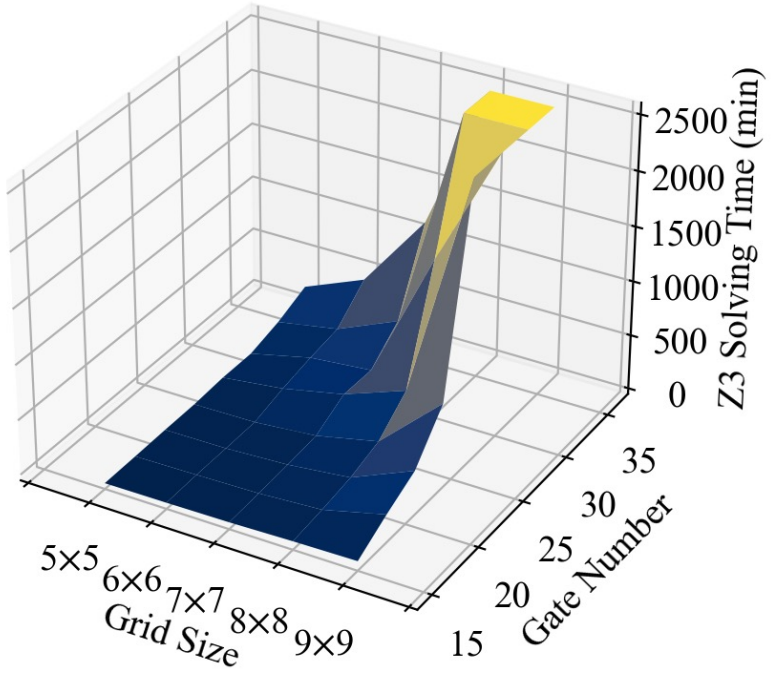
Insert SWAP gate to change the mapping



Coupling Graph of IBM QX 2

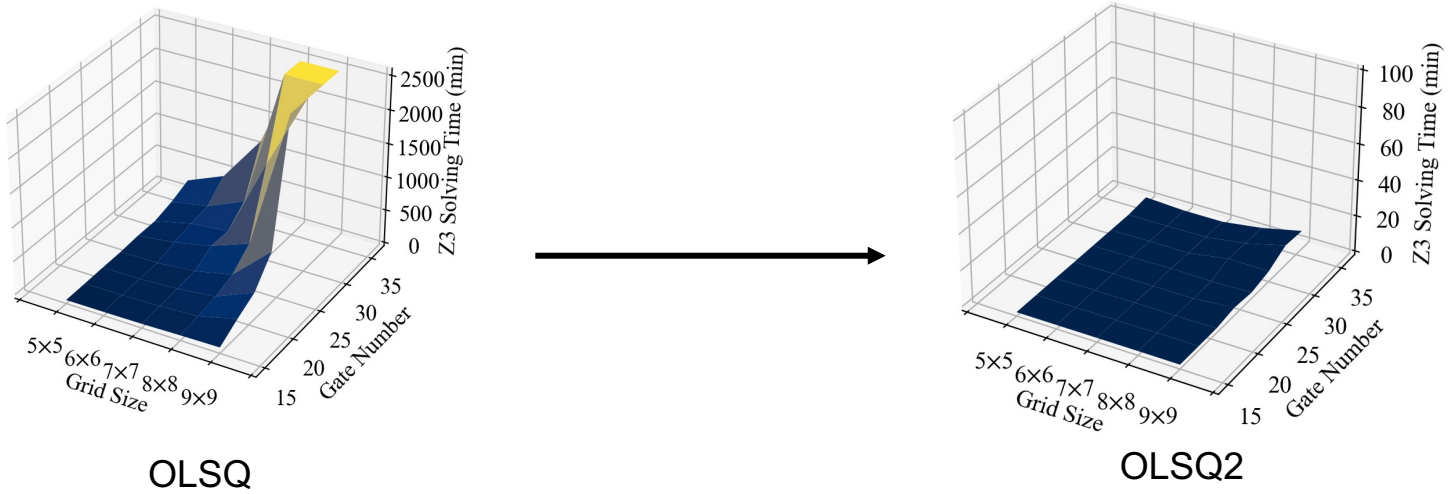
Leading Solution to Optimal Layout Synthesis (OLSQ): SMT-Based Approach

- Initial OLSQ design
  - Suboptimal variable encoding
  - Inefficient methods for SWAP and depth optimization



OLSQ 2

- Improve upon OLSQ:
  - Succinct formulation: Remove space variables
  - Better SMT encoding: Exploit bit vector
  - Efficient optimization methods: Apply incremental solving



OLSQ

OLSQ2

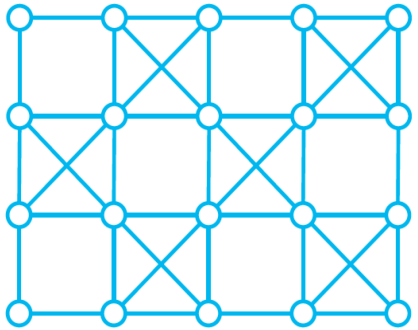
On average, achieve 387x speedup

* OLSQ2 is from [Lin et al., DAC'23]

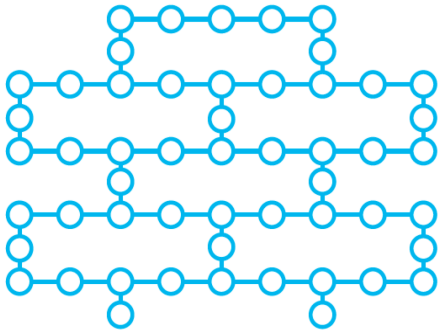
Current Focus: Support Diverse QC Platforms

- Superconducting devices
  - With different topologies

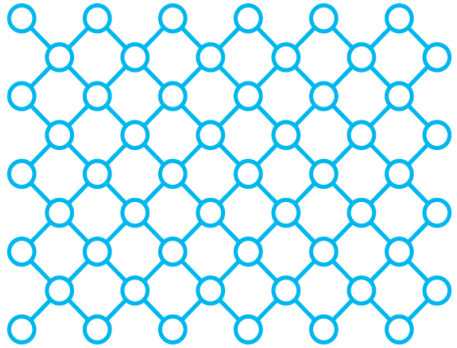
(b) Rigetti's Aspen-4 device graph



(c) IBM's Tokyo device graph

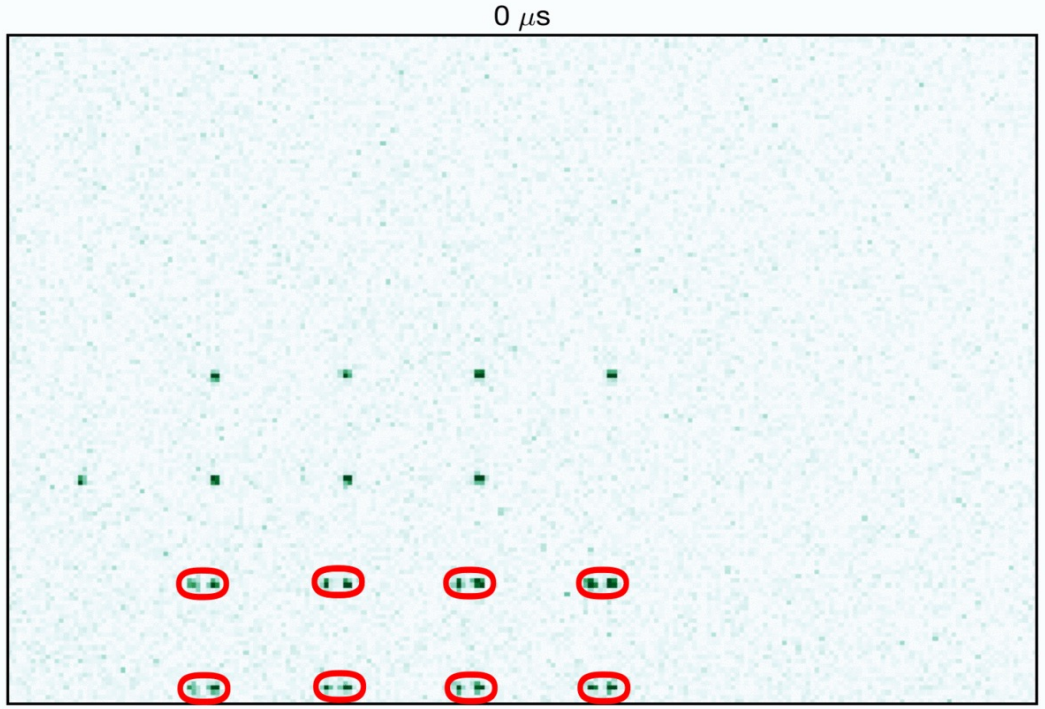


(d) IBM's Rochester device graph



(e) Google's Sycamore device graph

- Neutral Atom devices
  - Qubit movement



[Bluvstein et al., arXiv'21]

Both types can be supported using the SMT formulation (OLSQ2 and OLSQ-DRAA)

Concluding Remarks

- Most physical design researchers focus on the spatial domain,
- There is much value in considering the freedom in the time domain via scheduling to achieve better/simpler spatial solutions
- Ultimately, the EDA solution needs to decide the space and time coordinates of every computation
- **“Time and space are modes in which we think, not in which we exist”**

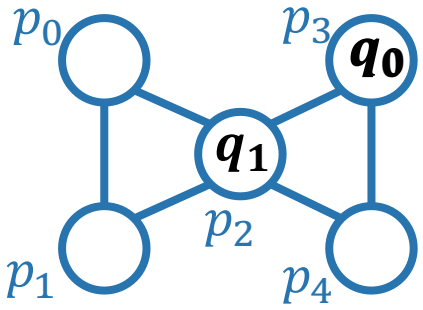
  -- Albert Einstein

Acknowledgement

- Prof. C. L. Liu, for countless valuable advice over the years
- The 3D IC physical design team at UCLA including Guojie Luo, Jie Wei, and Yang Zhang
- The HLS team at UCLA, including Yuze Chi, Yiping Fan, Licheng Guo, Guolin Han, Jerry Jiang, Jason Lau, Weikang Qiao, Linghao Song, Jie Wang, and their close collaborators at
  - Cornell: Ecenur Ustun, Xingyu Tian, and Zhiru Zhang
  - SFU: Zhenman Fang and Moazin Khatti
  - AMD/Xilinx: Alireza Kaviani, Chris Lavin, Pongstorn Maidee, and Yun Zhou
- The quantum layout synthesis team at UCLA, including Jason Kimko, Wan-Hsuan Lin, and Bochen Tan
- Taiwan NSC Travel Grant
- Last but certainly at least: Multi-decade friendship and collaboration with Martin Wong
  - Martin: Thank you and congratulations again!

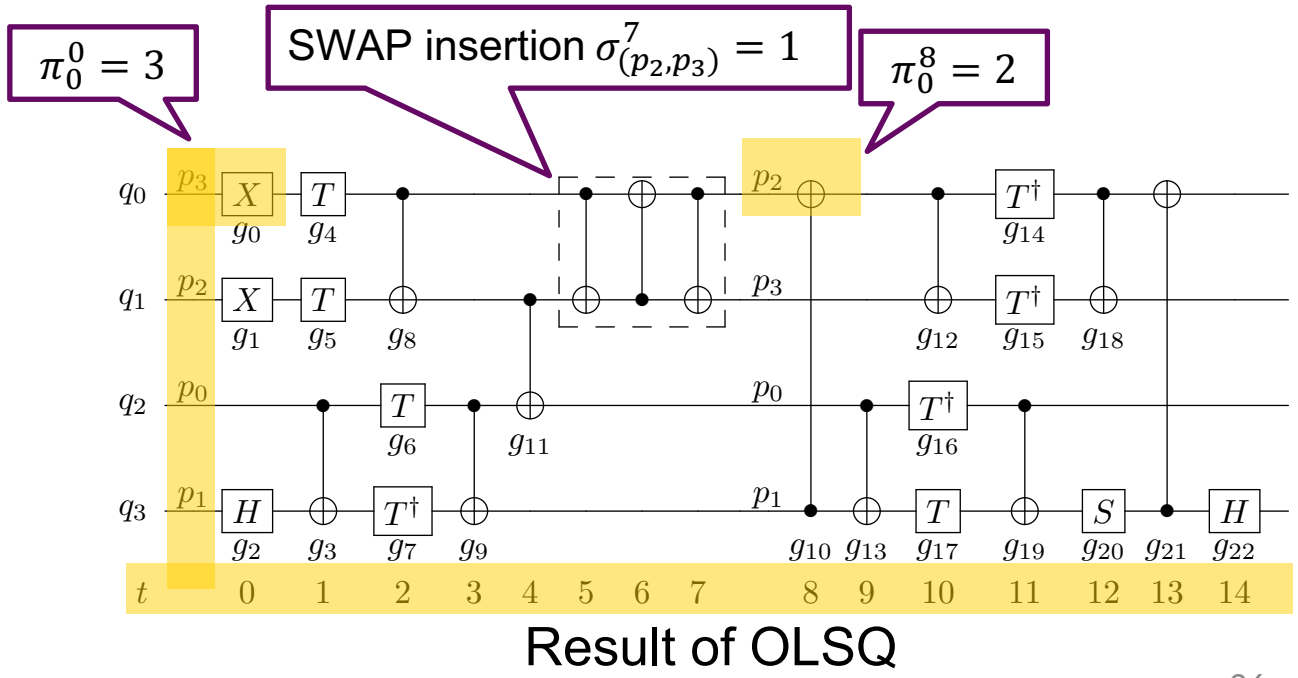
Thank You!

Leading Solution to Optimal Layout Synthesis: SMT-Based Approach



- Best Known Example: OLSQ
  - Spacetime Coordinate (t_l, x_l), for every gate g_l: $t_l = t$ and $x_l = k$ iff. g_l is executed at time t and qubit/edge k
  - Mapping π_q^t: at time t, logical qubit q is mapped to the physical qubit π_q^t
  - Use of SWAP σ_e^t: $\sigma_e^t = 1$ iff. there is a SWAP on edge e and its last time step is t

  - Example constraints:
    - Mapping injectivity: $\pi_0^t \neq \pi_1^t$
    - Gate dependency: $g_0 < g_4$
    - Consistency between mapping and space-time variables
    - Valid SWAP insertion



Result of OLSQ

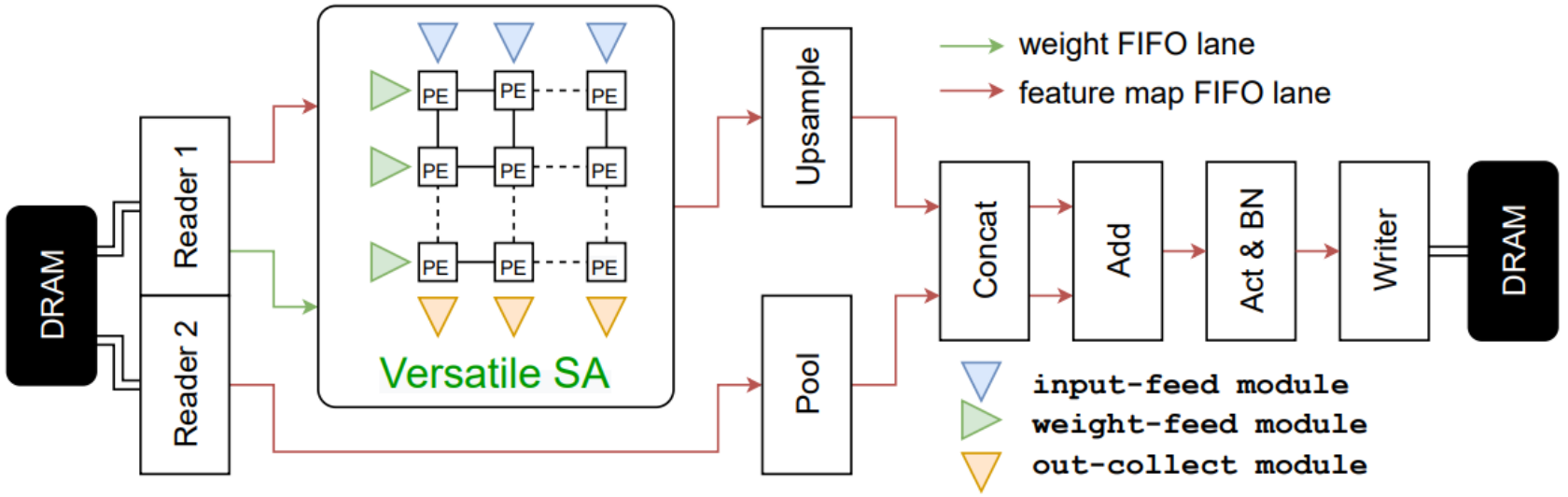
* OLSQ is from [Tan and Cong, ICCAD'20]

Case Study 2: FlexCNN Using TAPA

- **FlexCNN: an end-to-end automated DNN synthesis framework**
- **From ONNX to bitstream on FPGAs**

Module	Lines of Code	Code Generation
Reader 1	1,046	Template-based
Reader 2	446	Template-based
Systolic Array	4,801	Automatic
Pool	254	Template-based
Upsample	221	Template-based
Concat	350	Template-based
Add	314	Template-based
Act & BN	320	Template-based
Writer	824	Template-based
Top	6,292	Automatic
Total	**14,868**	

FlexCNN without TAPA	FlexCNN with TAPA
Fails Placement & Route	Achieves up to 266 MHz



Large dataflow design composed using TAPA