Design Space Exploration for Power-Efficient Mixed-Radix Ling Adders

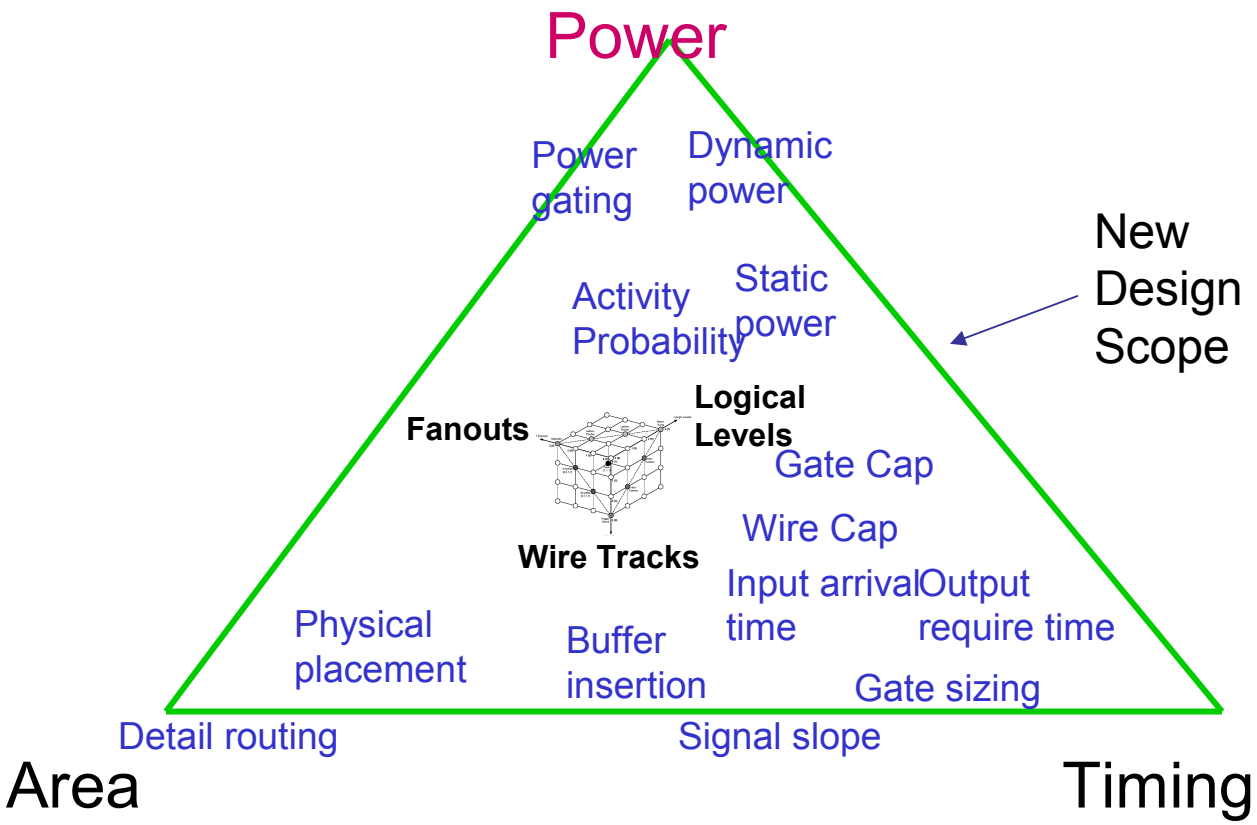
Chung-Kuan Cheng
Computer Science and Engineering
Depart. University of California, San Diego

Outline

- Prefix Adder Problem
  - Background & Previous Work
  - Extensions: High-radix, Ling
- Our Work
  - Area/Timing/Power Models
  - Mixed-Radix (2,3,4) Adders
  - ILP Formulation
- Experimental Results
- Future Work

Prefix Adder – Challenges

- Increasing impact of physical design and concern of power.



Binary Addition

- **Input:** two *n*-bit binary numbers $a_{n-1}\ldots a_1 a_0$ and $b_{n-1}\ldots b_1 b_0$, one bit carry-in c_0
- **Output:** n-bit sum $s_{n-1}\ldots s_1 s_0$ and one bit carry out c_n
- Prefix Addition: Carry generation & propagation

$$\text{Generate}: \quad g_i = a_i b_i$$

$$\text{Propagate}: \quad p_i = a_i \oplus b_i$$

$$c_{i+1} = g_i + p_i \cdot c_i$$

$$s_i = c_i \oplus (a_i \oplus b_i)$$

Prefix Addition – Formulation

Pre-processing:

$$g_i = a_i b_i \qquad p_i = a_i \oplus b_i$$

Prefix Computation:

$$G_{[i:k]} = G_{[i:j]} + P_{[i:j]} G_{[j-1:k]}$$

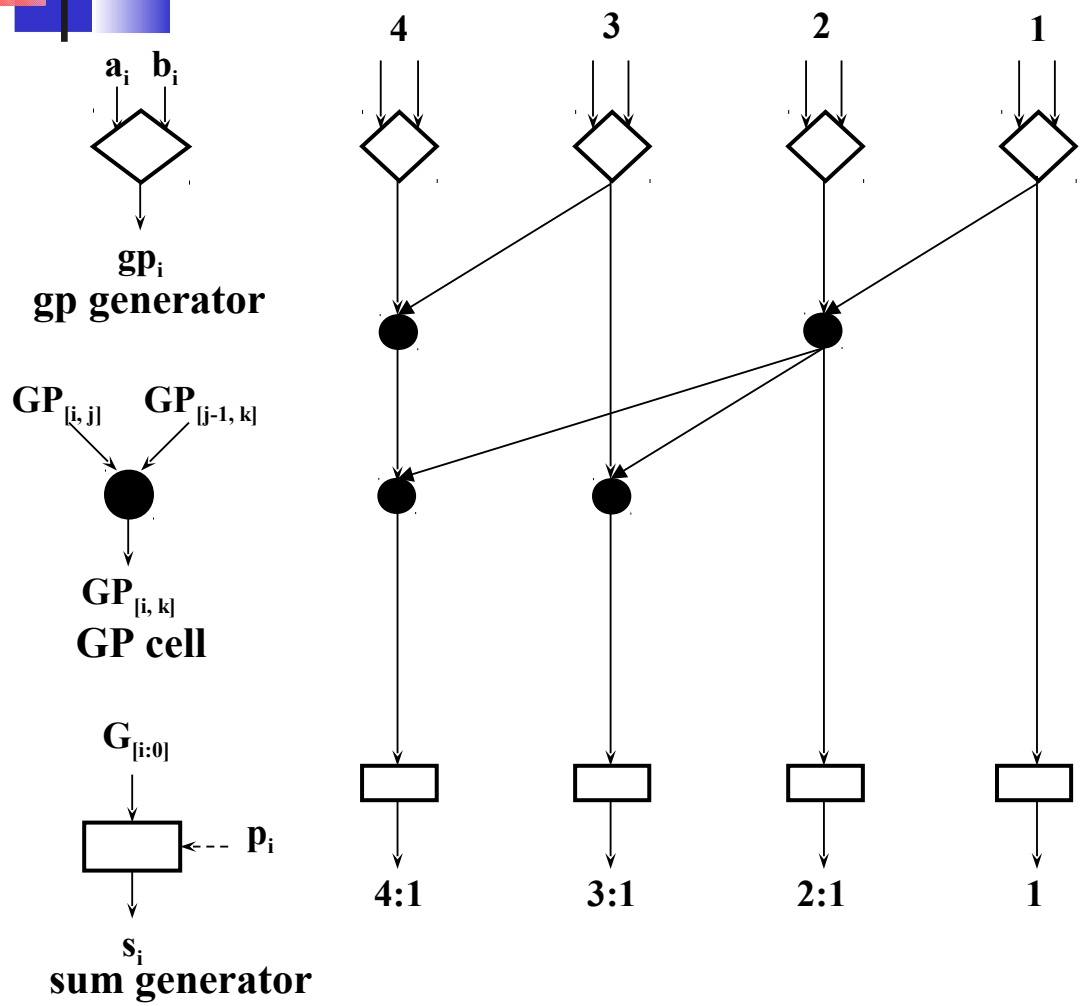
$$P_{[i:k]} = P_{[i:j]} P_{[j-1:k]}$$

Post-processing:

$$c_{i+1} = G_{[i:0]} + P_{[i:0]} \cdot c_0$$

$$s_i = p_i \oplus c_i$$

Prefix Adder – Prefix Structure Graph

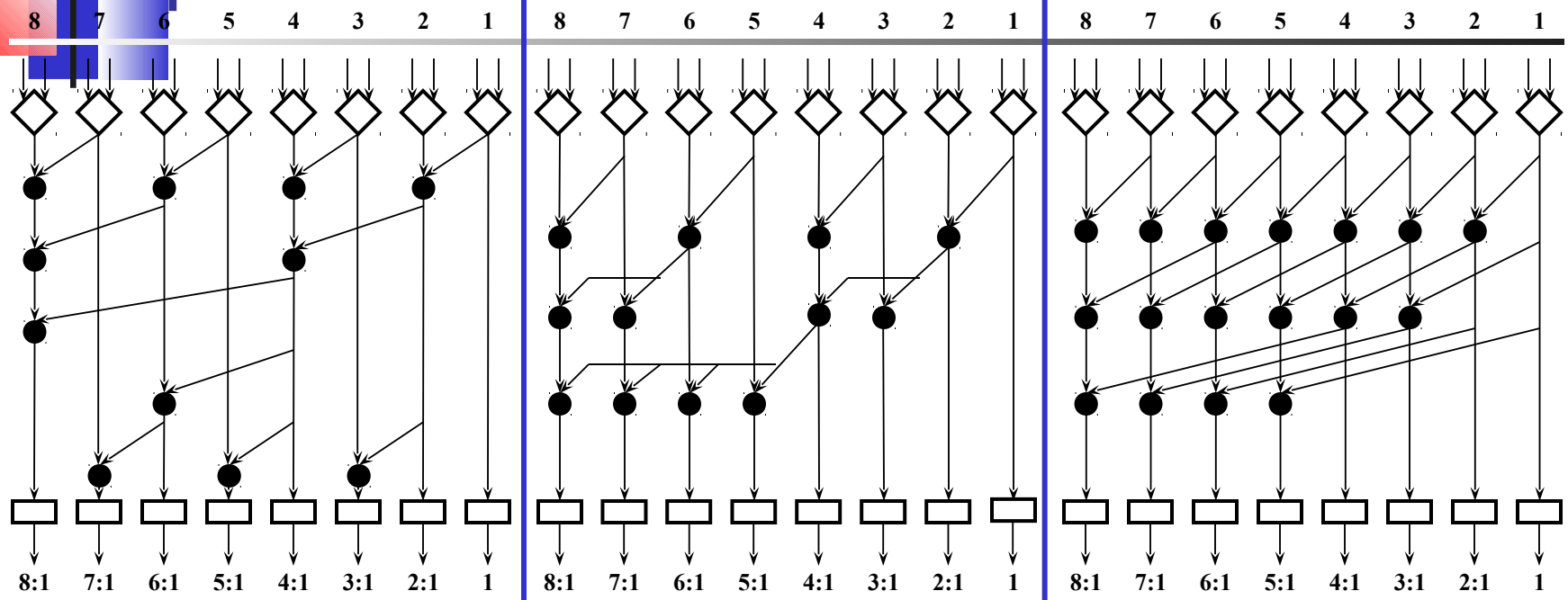


Pre-processing

Prefix Computation

Post-processing

Previous Works – Classical prefix adders



Brent-Kung:
Logical levels: $2\log_2 n - 1$
Max fanouts: 2
Wire tracks: 1

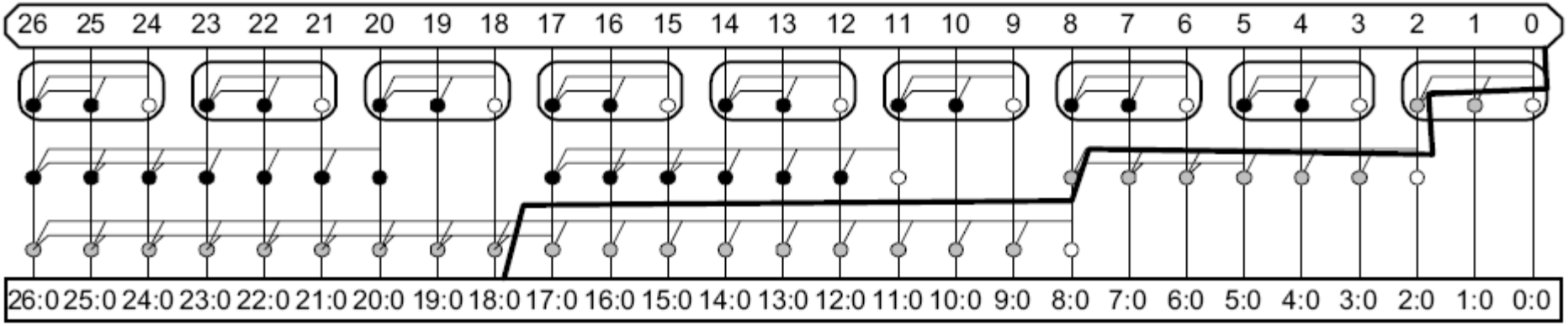
Sklansky:
Logical levels: $\log_2 n$
Max fanouts: n/2
Wire tracks: 1

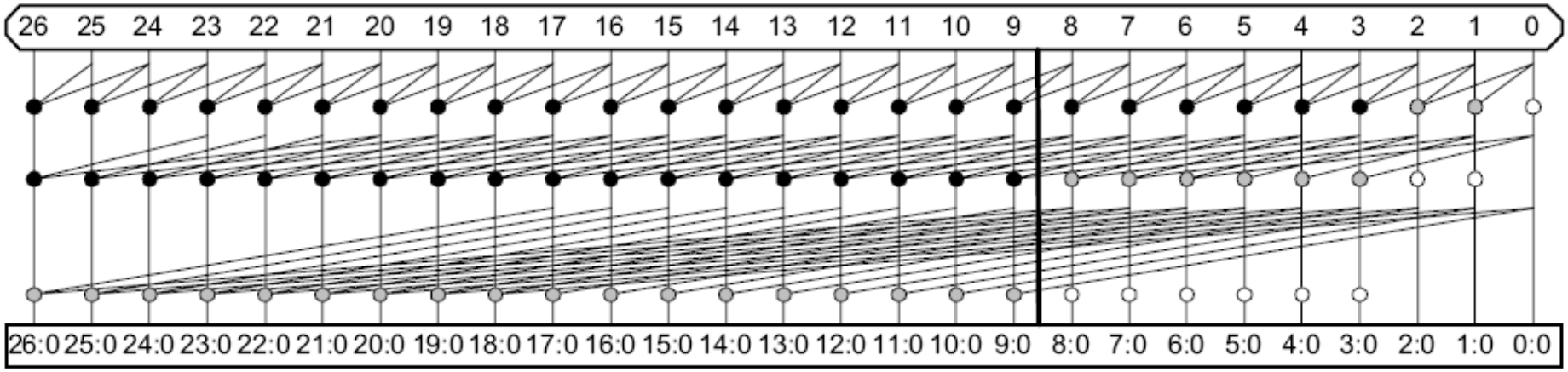
Kogge-Stone:
Logical levels: $\log_2 n$
Max fanouts: 2
Wire tracks: n/2

High-Radix Adders

- Each cell has more than two fan-in's
- Pros: less logic levels
  - 6 levels (radix-2) vs. 3 levels (radix-4) for 64-bit addition
- Cons: larger delay and power in each cell

Radix-3 Sklansky & Kogge-Stone Adder





David Harris, "Logical Effort of Higher Valency Adders"

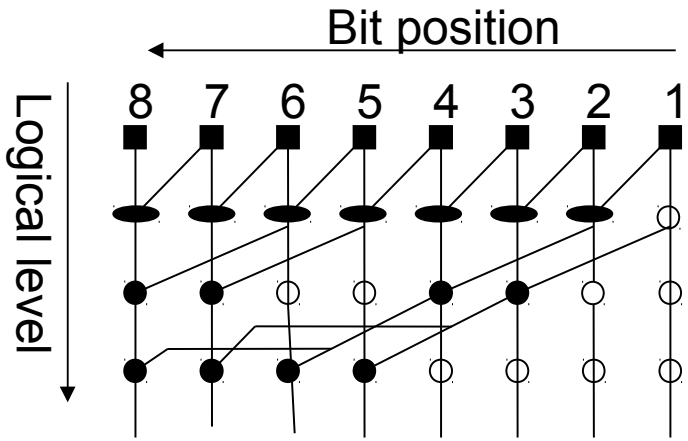
Ling Adders

	Prefix	Ling
Pre-processing:	$g_i = a_i b_i \qquad p_i = a_i \oplus b_i$	$g_i = a_i b_i,\ p_i = a_i + b_i$ $t_i = a_i \oplus b_i$
Prefix Computation:	$G_{[i:k]} = G_{[i:j]} + P_{[i:j]} G_{[j-1:k]}$ $P_{[i:k]} = P_{[i:j]} P_{[j-1:k]}$	$G^*_{[i:i-1]} = g_i + g_{i-1},\ P^*_{[i:i-1]} = p_i \bullet p_{i-1}$ $G^*_{[i:k]} = G^*_{[i:j]} + P^*_{[i-1:j-1]} G^*_{[j-1:k]}$ $P^*_{[i:k]} = P^*_{[i:j]} P^*_{[j-1:k]}$
Post-processing:	$c_i = G_{[i-1:0]} + P_{[i-1:0]} \cdot c_0$ $s_i = p_i \oplus c_i$	$c_i = p_{i-1} \cdot G^*_{[i-1:0]}$ $s_i = \overline{G^*_{[i-1:0]}} \times t_i + G^*_{[i-1:0]} \times (t_i \oplus p_{i-1})$

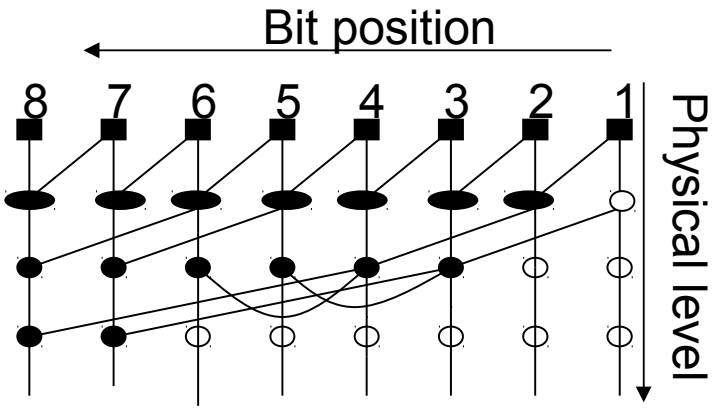
An 8-bit Ling Adder

Area Model

- Distinguish physical placement from logical structure, but keep the bit-slice structure.



Logical view

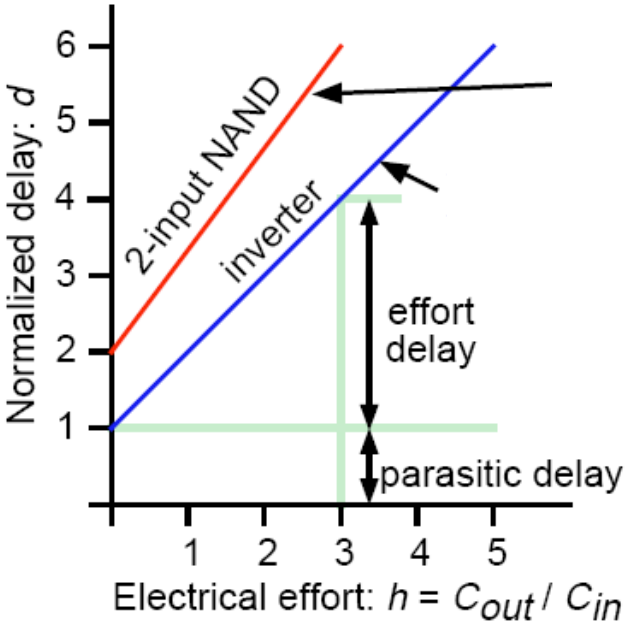


Physical view

Compact placement

Timing Model

- Cell delay calculation:



$$d = f + p$$

Effort Delay

Intrinsic Delay

$$f = g \bullet h$$

Logical Effort

Electrical Effort = Cout/Cin = (fanouts+wirelength) / size

Intrinsic properties of the cell

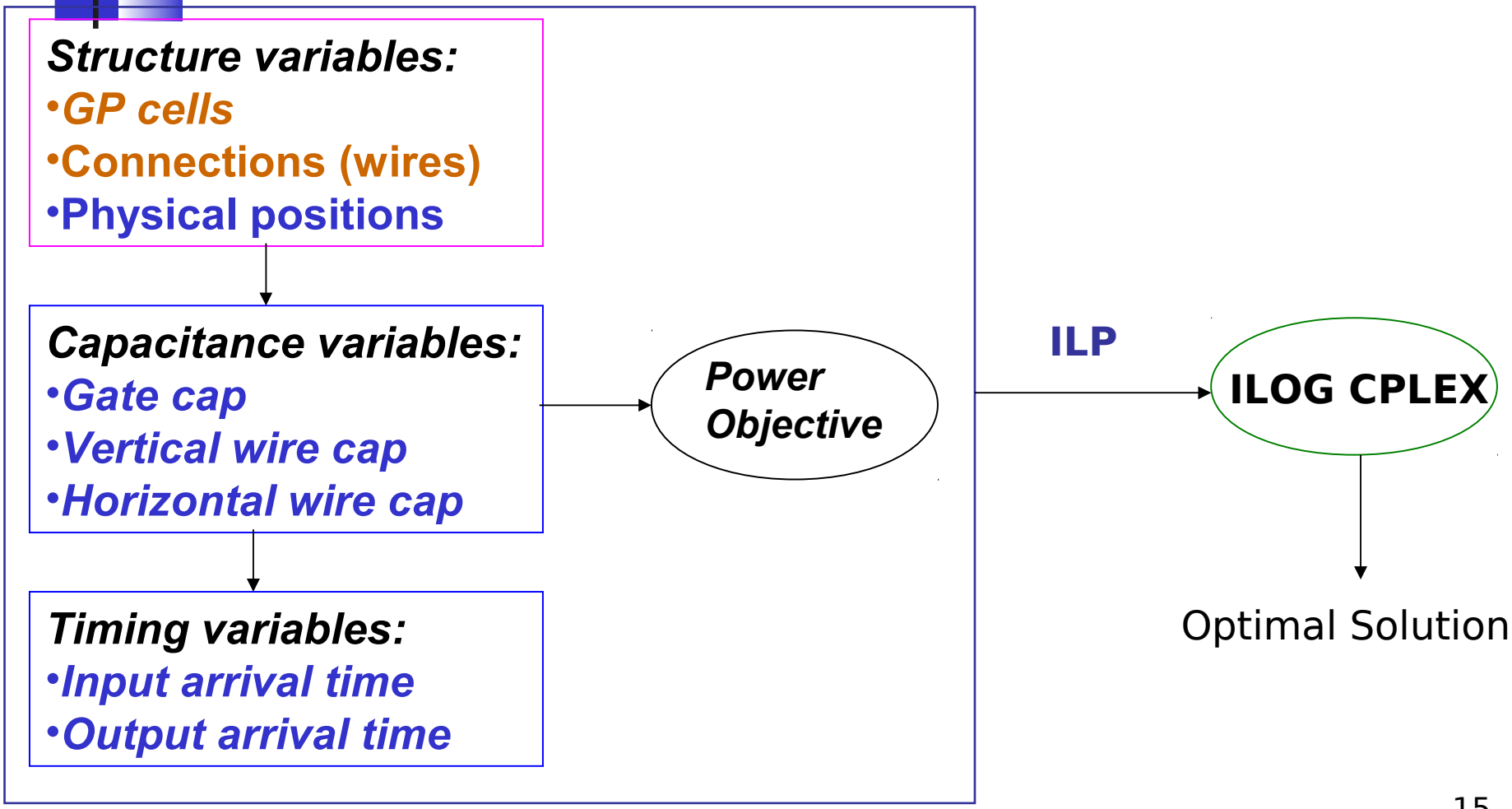
Power Model

- Total power consumption:
  Dynamic power + Static Power
- Static power: leakage current of device
  $P_{sta} = \lambda * \#cells$
- Dynamic power: current switching capacitance
  $P_{dyn} = \rho \times C_{load}$
- ρ is the switching probability
  $\rho = j$ (j is the logical level*)

$$P_{total} = P_{dyn} + P_{sta} = j \cdot C_{load} + \lambda \cdot \#cells$$

* Vanichayobon S, etc, “Power-speed Trade-off in Parallel Prefix Circuits”

ILP Formulation Overview

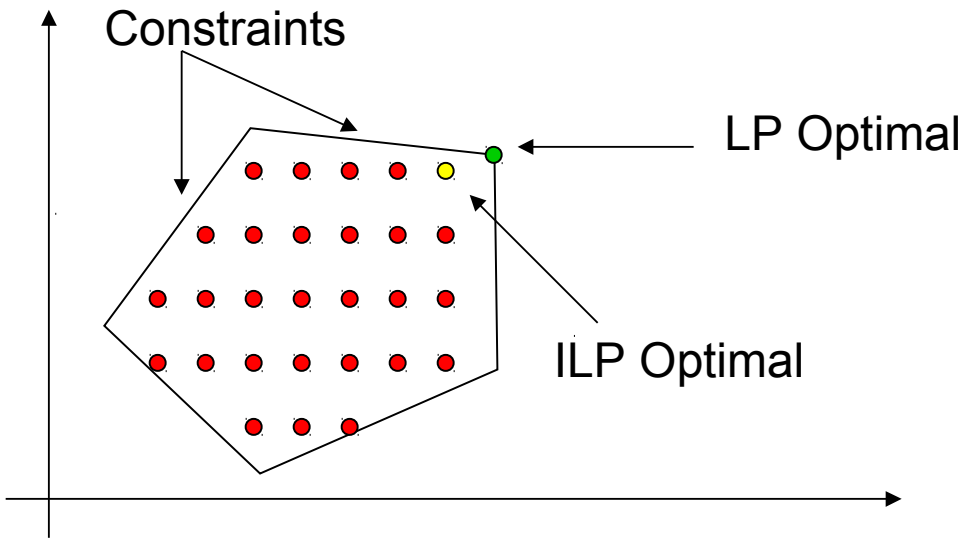


Integer Linear Programming (ILP)

- **ILP:** Linear Programming with integer variables.
- Difficulties and techniques:
  - Constraints are not linear
    - Linearize using pseudo linear constraints
  - Search Space too large
    - Reduce search space
  - Search is slow
    - Add redundant constraints to speedup

ILP – Integer Linear Programming

- **Linear Programming:** linear constraints, linear objective, fractional variables.
- **Integer Linear Programming:** Linear Programming with integer variables.



ILP – Pseudo-Linear Constraint

- *A constraint is called pseudo-linear if it's not effective until some integer variables are fixed.*

Problem:

Minimize: x_3

Subject to: $x_1 \geq 300$

$x_2 \geq 500$

$x_3 = \min(x_1, x_2)$

LP objective: 0
ILP objective: 300

ILP formulation:

Minimize: x_3

Subject to: $x_1 \geq 300$

$x_2 \geq 500$

$x_3 \leq x_1$

$x_3 \leq x_2$

$x_3 \geq x_1 - 1000\, b_1$ (1)

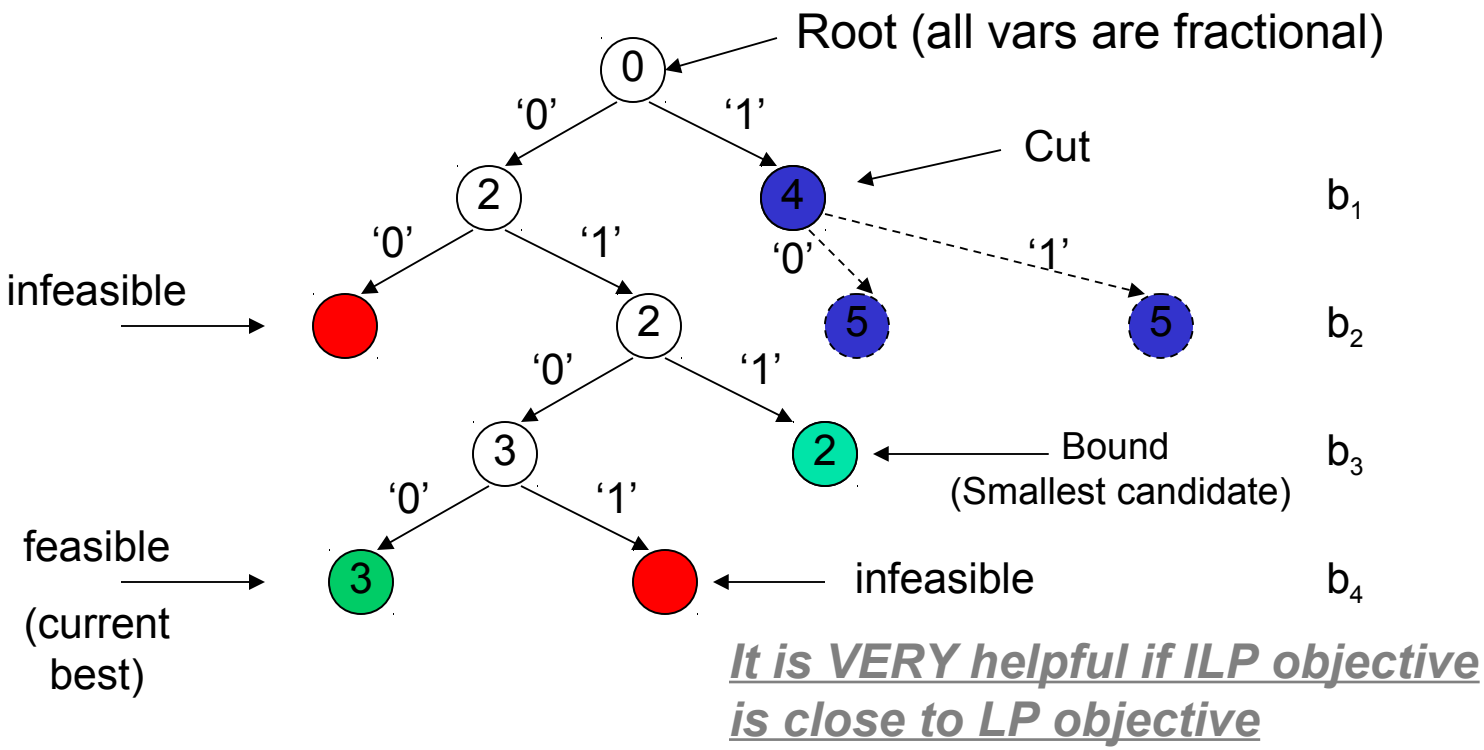
$x_3 \geq x_2 - 1000\,(1 - b_1)$ (2)

b_1 *is binary*

- *Pseudo-linear constraints mostly arise from IF/ELSE scenarios*
  - *binary decision variables are introduced to indicate true or false.*

ILP Solver Search Procedure

Minimize $F(b_1, b_2, b_3, b_4, f_1, \ldots)$ **b_i is binary**



It is VERY helpful if ILP objective is close to LP objective

Interval Adjacency Constraint

(column id, logic level)

Linearization for Interval Adjacency Constraint

$$y^R_{(i,h)} = y^L_{(k,l)} + 1 \quad \text{if} \quad wl(i,j,h) = wr1(i,j,k,l) = 1$$

$$[y^L_{(i,h)}, \boxed{y^R_{(i,h)}}]\ [y^L_{(k1,l1)}, y^R_{(k1,l1)}][y^L_{(k2,l2)}, y^R_{(k2,l2)}]$$

Left interval bound equal to column index

$$y^L_{(i,j)} = i$$

$$y^R_{(i,h)} = \sum_{(k,l)} k \cdot wr1(i,j,k,l) + 1 \quad \text{if} \quad wl(i,j,h) = 1$$

Linearize

$$y^R_{(i,h)} \geq \sum_{(k,l)} k \cdot wr1(i,j,k,l) - \boxed{n \cdot (1 - wl(i,j,h))} + 1$$

$$y^R_{(i,h)} \leq \sum_{(k,l)} k \cdot wr1(i,j,k,l) + \boxed{n \cdot (1 - wl(i,j,h))} + 1$$

$$[y^L_{(i,j)}, y^R_{(i,j)}]$$

Pseudo Linear

Search Space Reduction

- Ling's adder: separate odd and even bits
- Double the bit-width we are able to search

Redundant Constraints

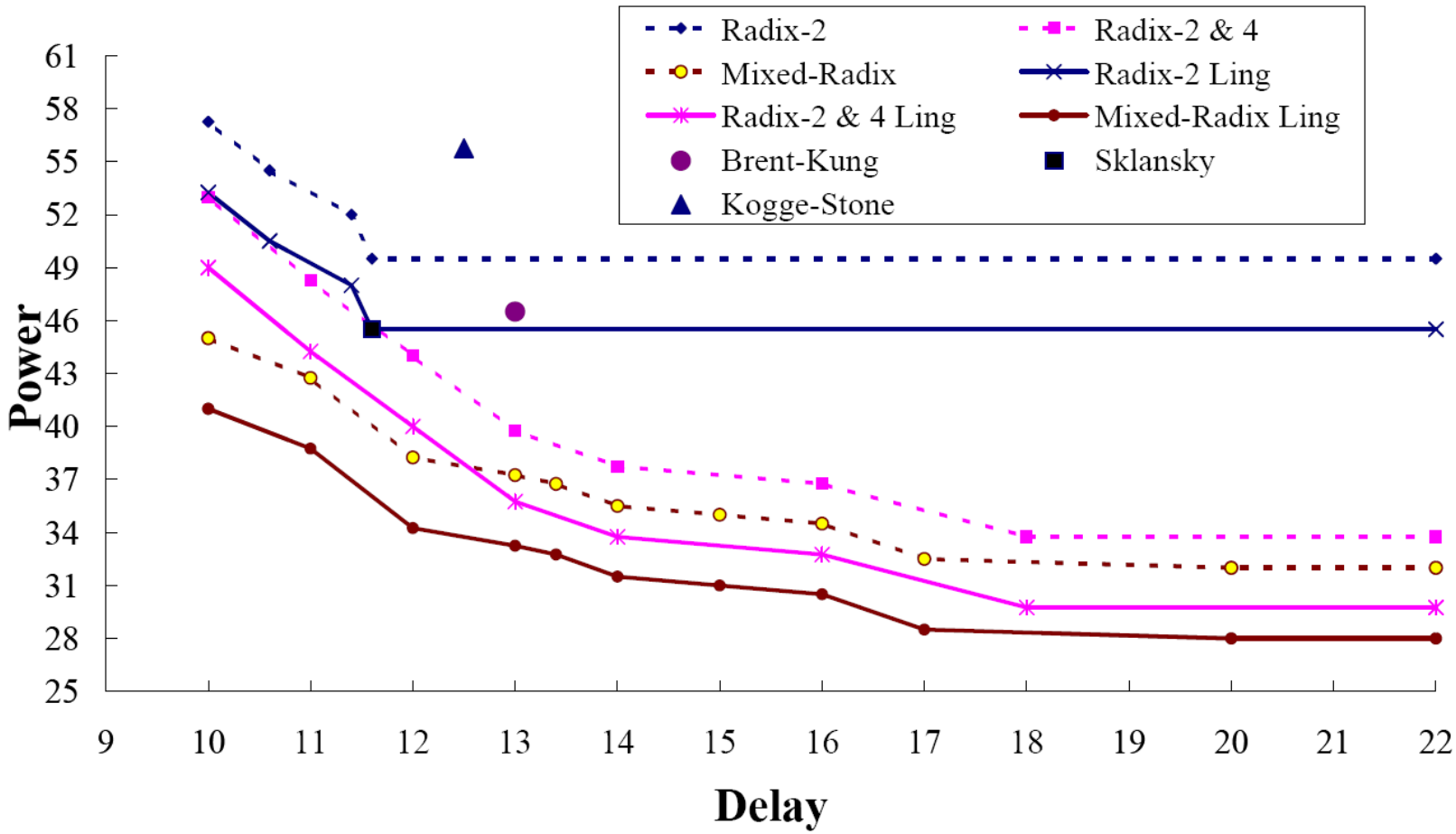
- Cell *(i,j)* is known to have logic level *j* before wire connection
- Assume load is *MinLoad* (fanout=1 with minimum wire length):

$$P_{(i,j)} \geq j \cdot MinLoad + \lambda$$

- Cell (i,j) has a path of length *j-1*
- Assume each cell along the path has *MinLoad*

$$T_{(i,j)} \geq j \cdot (PD + LE \cdot MinLoad)$$

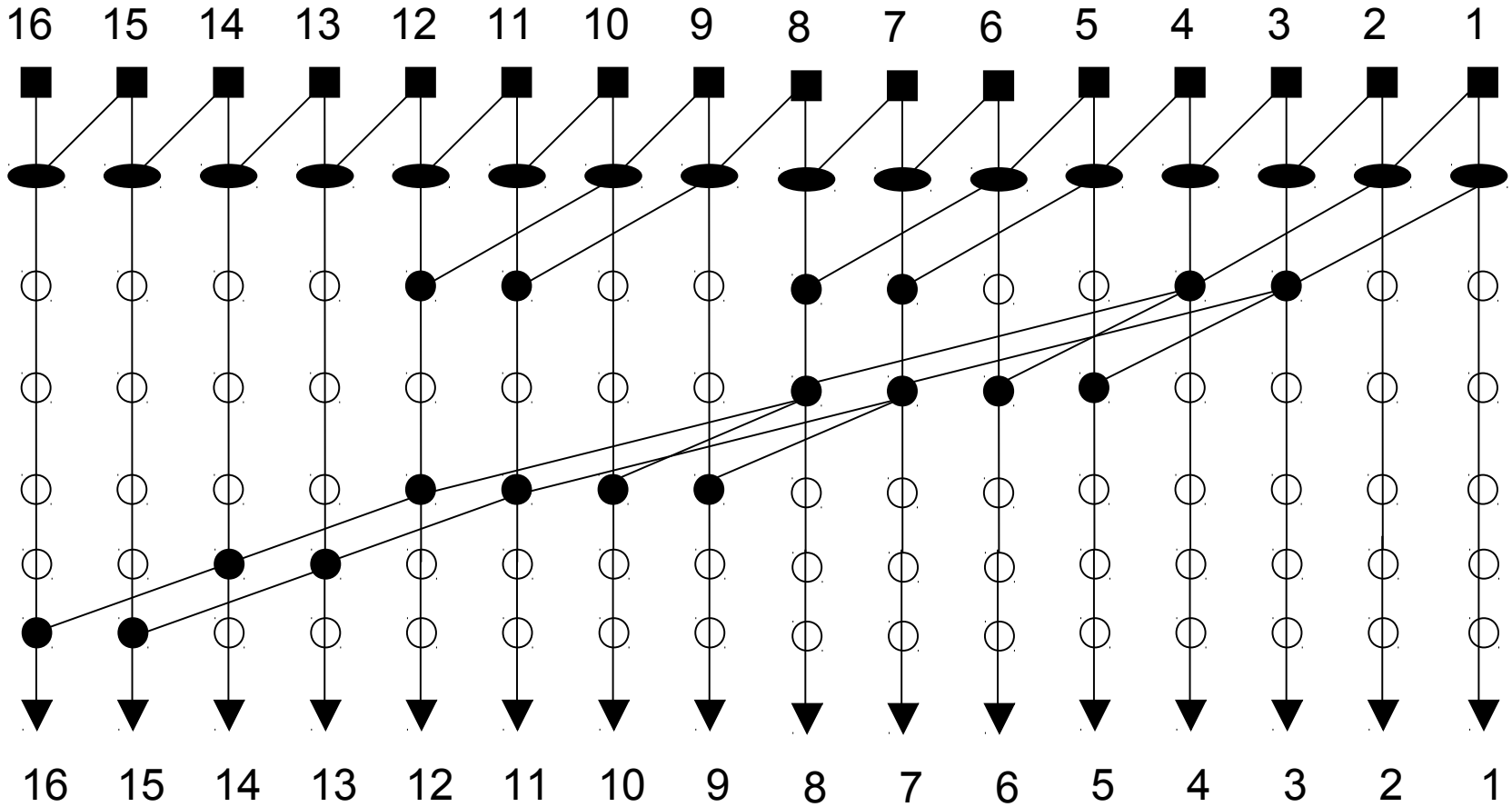
Experiments – 16-bit Uniform Timing



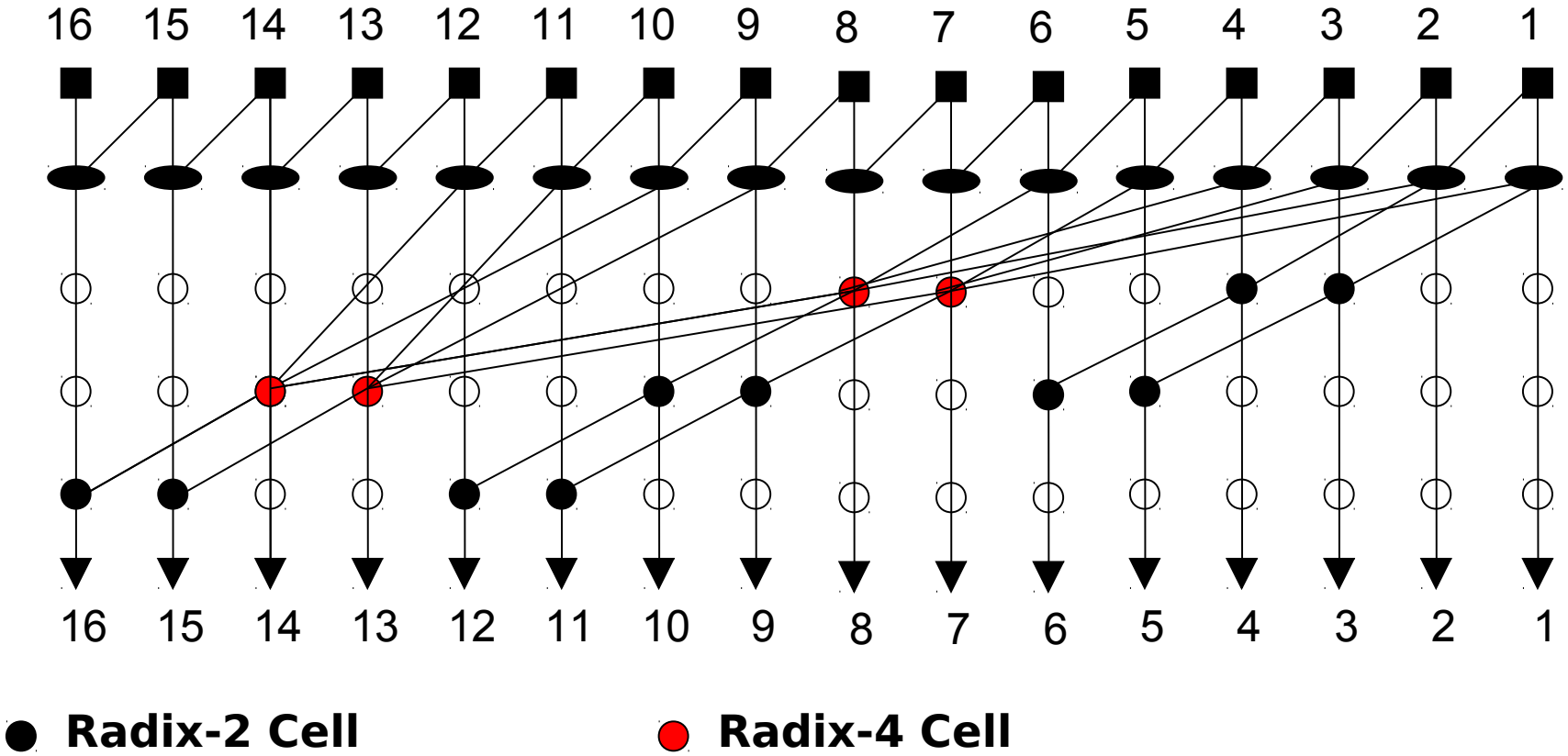
Experiments – 16-bit Uniform Timing

Radix	Delay (D_{FO4})	Power (P_{FO4})	CPU (sec)	Radix	Delay (D_{FO4})	Power (P_{FO4})	CPU (sec)
2	11.6	45.5	1	2,3,4	20	28	10
2	11.4	48	1	2,3,4	17	28.5	21
2	10.6	50.5	3	2,3,4	16	30.5	68
2	10	53.25	11	2,3,4	15	31	125
				2,3,4	14	31.5	200
2,4	18	29.75	10	2,3,4	13	33.25	541
2,4	16	32.75	67	2,3,4	12	34.25	2850
2,4	14	33.75	232	2,3,4	11	38.75	5647
2,4	13	35.75	613	2,3,4	10	41	71687
2,4	12	40	1806	B-K	15	41.5	-
2,4	11	44.25	6187	Sklansky	11	45.5	-
2,4	10	49	32576	K-G	12.5	55.75	-

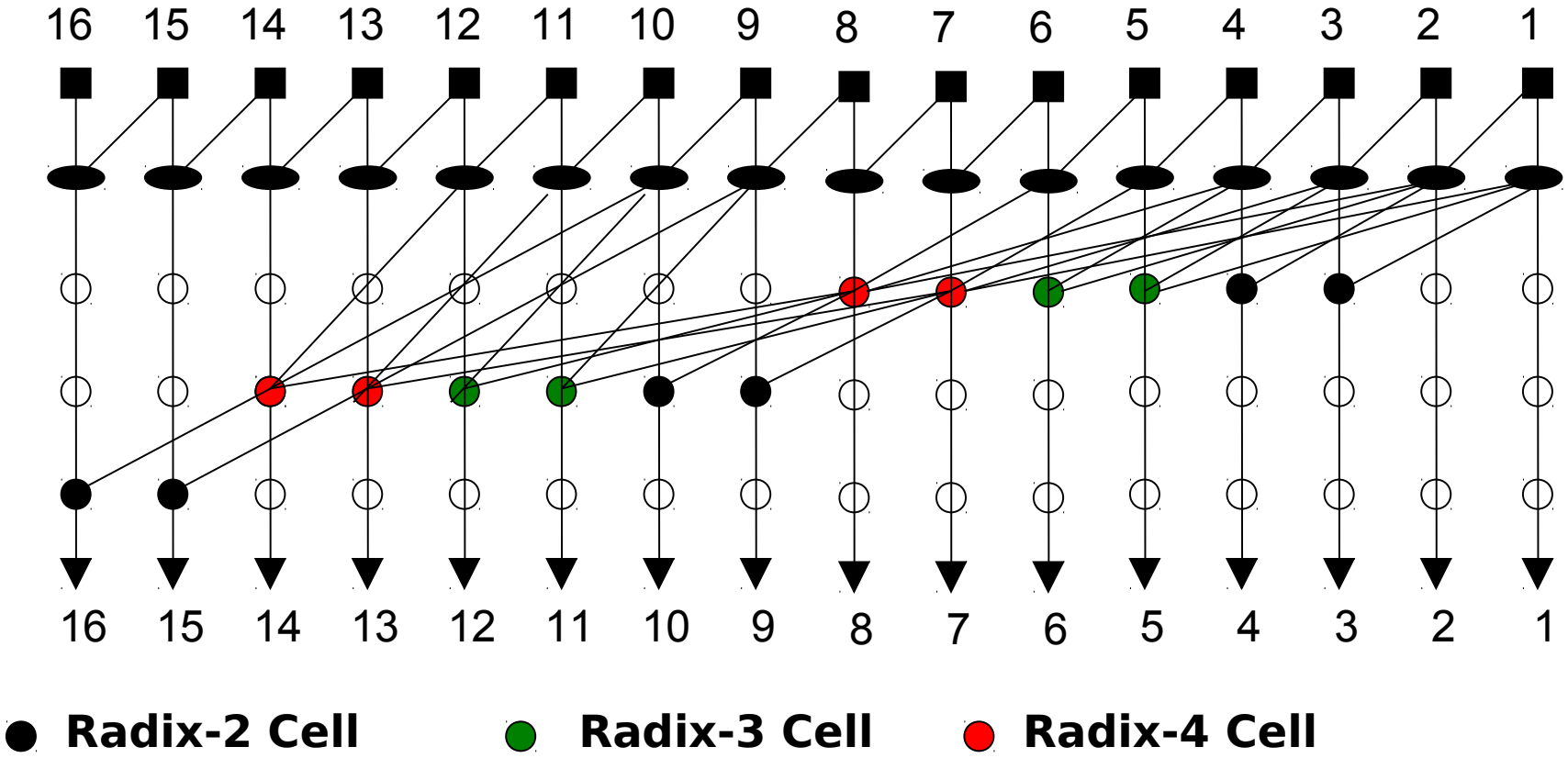
Min-Power Radix-2 Adder
(delay= 22, power = 45.5FO4)



Min-Power Radix-2&4 Adder (delay=18, power = 29.75FO4)



Min-Power Mixed-Radix Adder (delay=20, power = 28.0FO4)



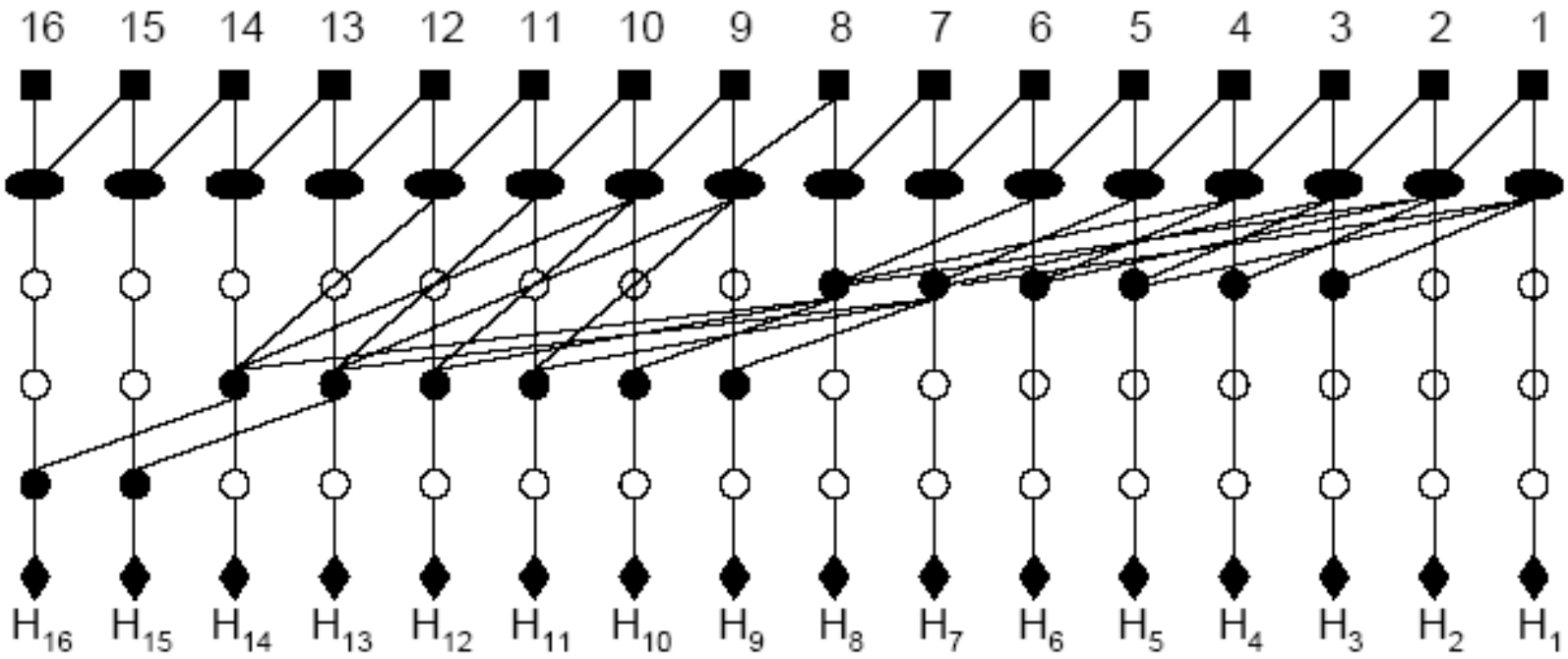
Experiments – 16-bit Non-uniform Time (Mixed Radix)

Case	Power (Prefix) (P_{FO4})	Power (Ling) (P_{FO4})	Improvement
Increasing Arrival Time	35.5	27.0	23.9%
Decreasing Arrival Time	34.5	30.5	11.6%
Convex Arrival Time	35.9	32.4	9.7%
Increasing Required Time	34.5	30.5	11.6%
Decreasing Required Time	36.5	32.5	11.0%
Convex Required Time	36.5	32.5	11.0%

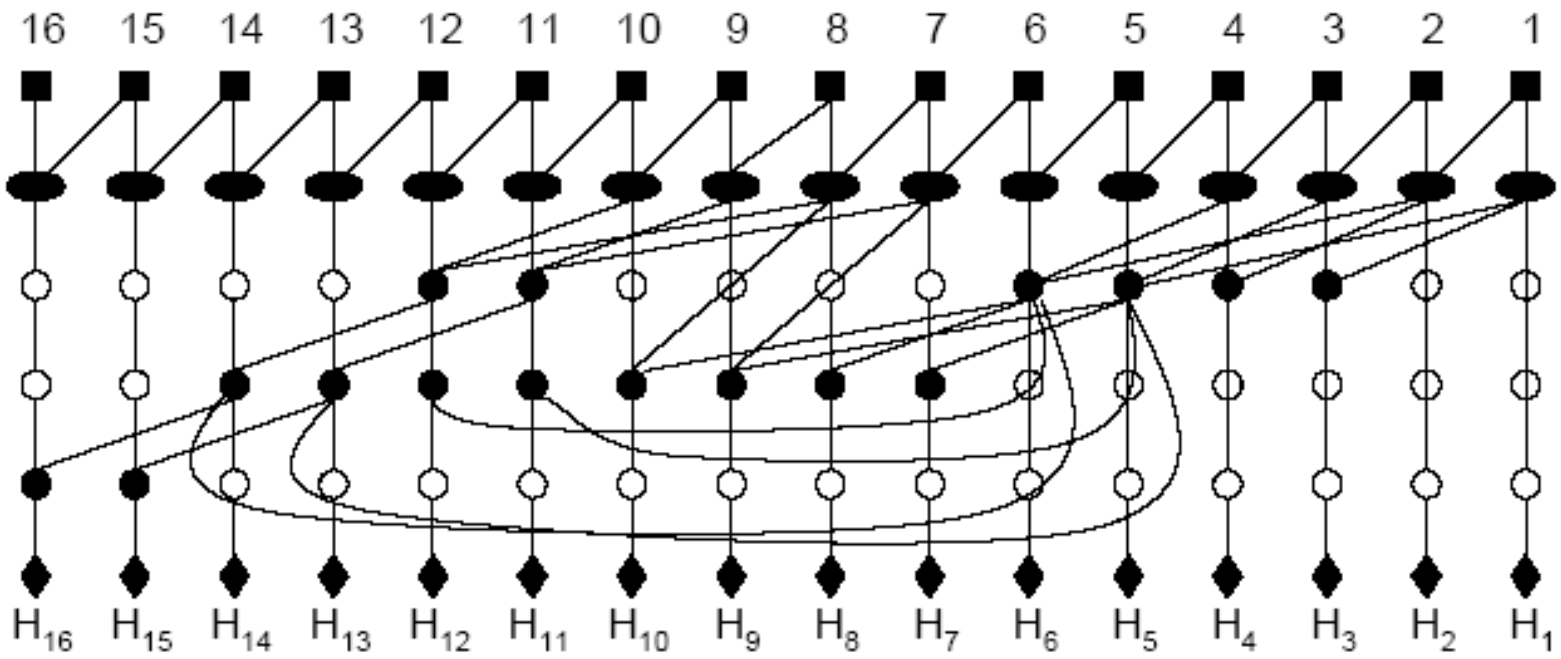
ILP is able to handle non-uniform timings

Ling adders are most superior in increasing arrival time
– faster carries

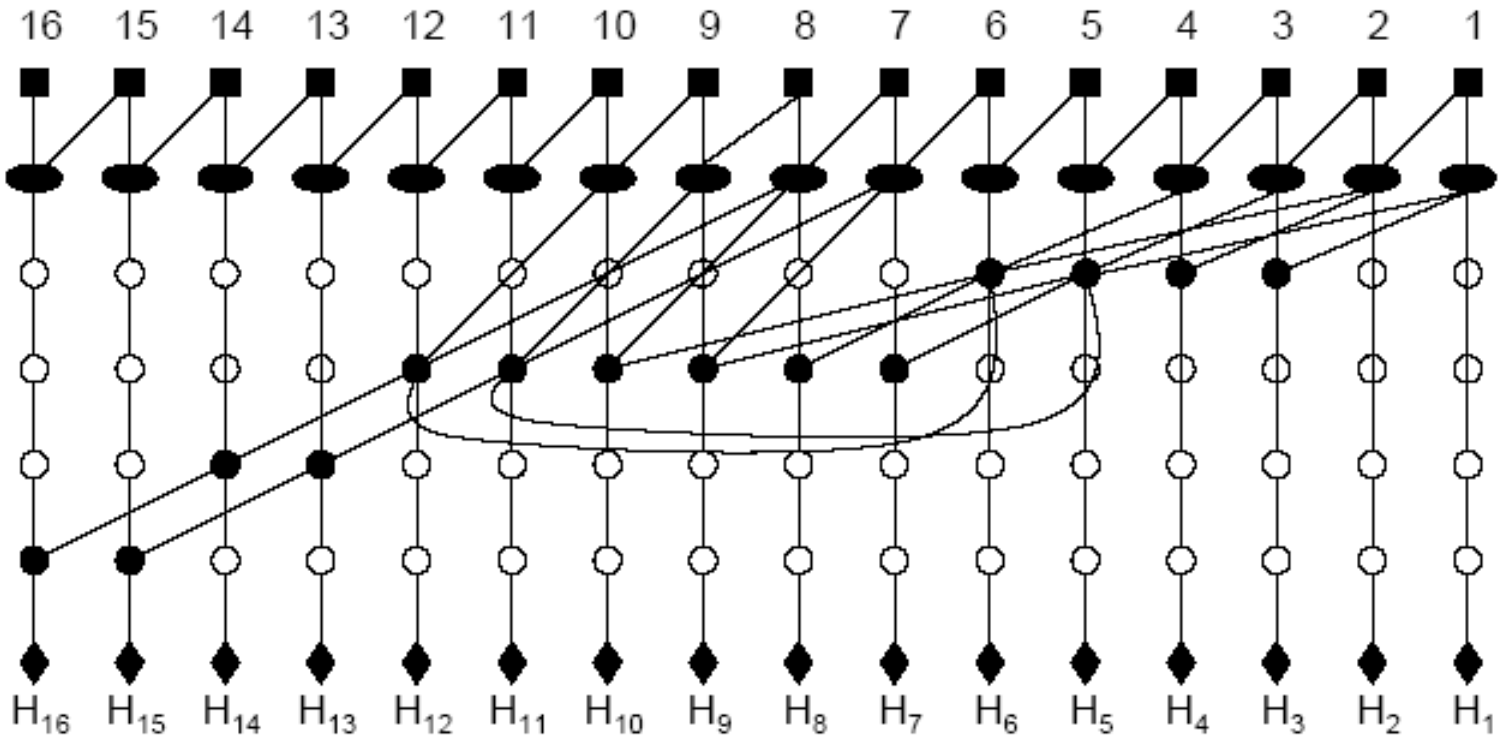
Increasing Arrival Time
(delay=35.5, power = 27.0FO4)



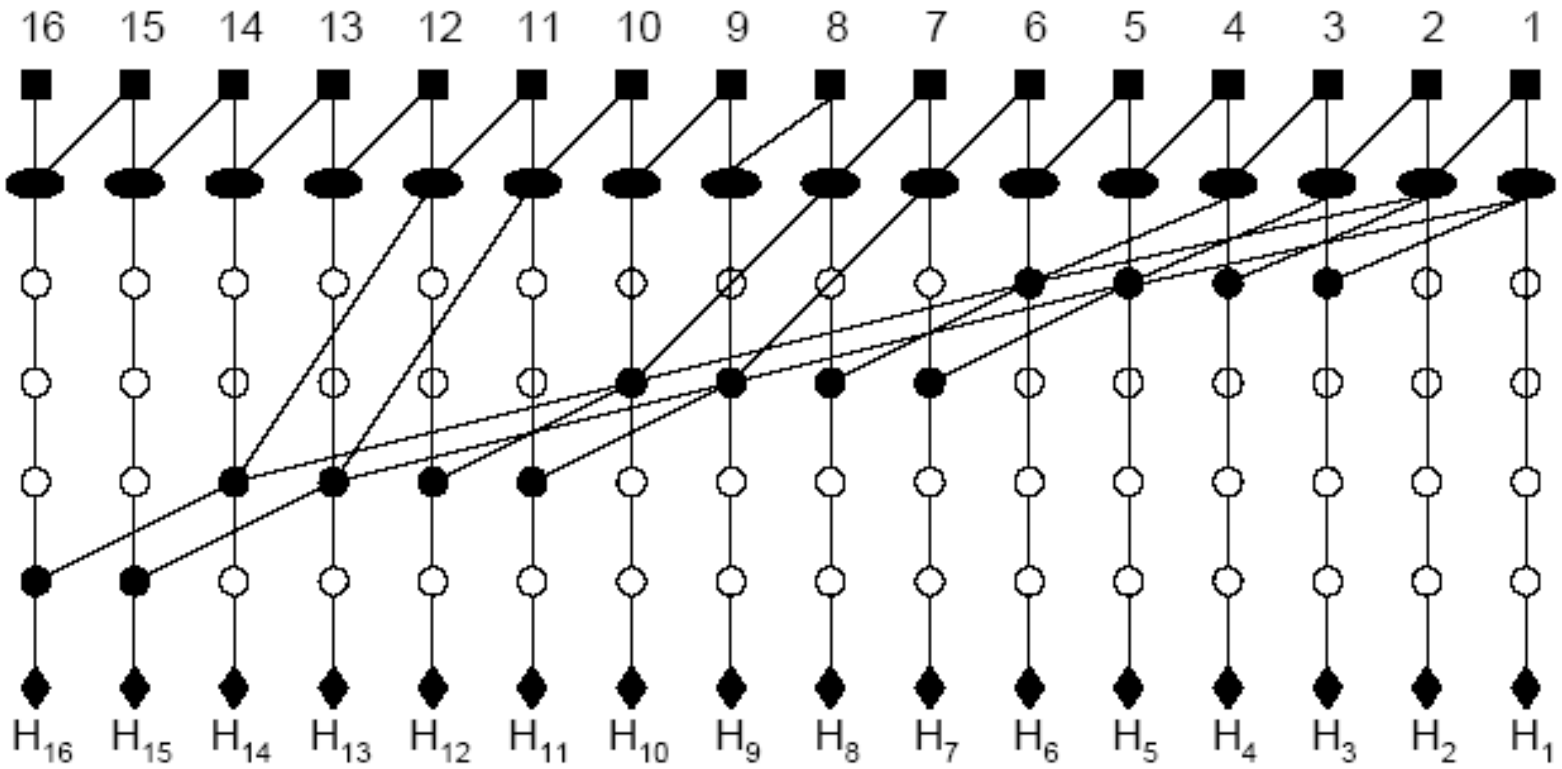
Decreasing Arrival Time
(delay=34.5, power = 30.5FO4)



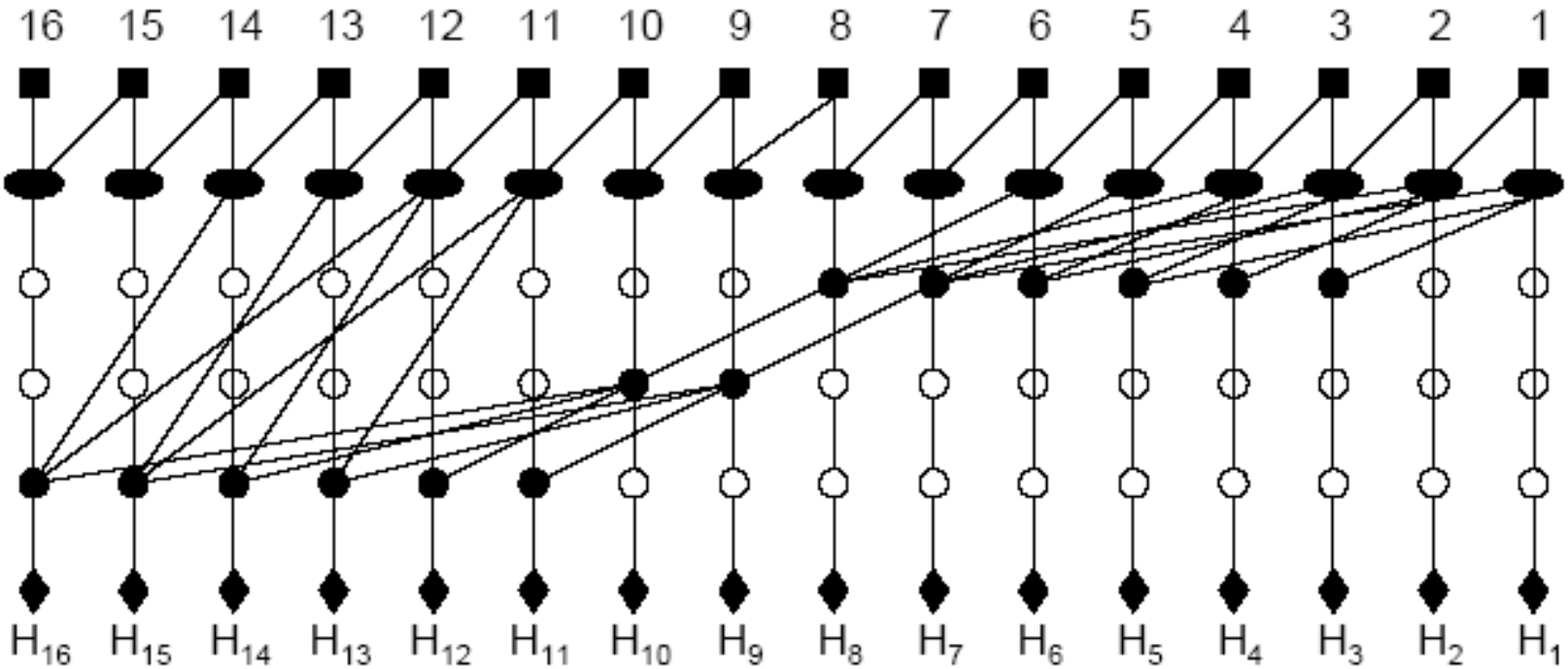
Convex Arrival Time
(delay=35.9, power = 32.4FO4)



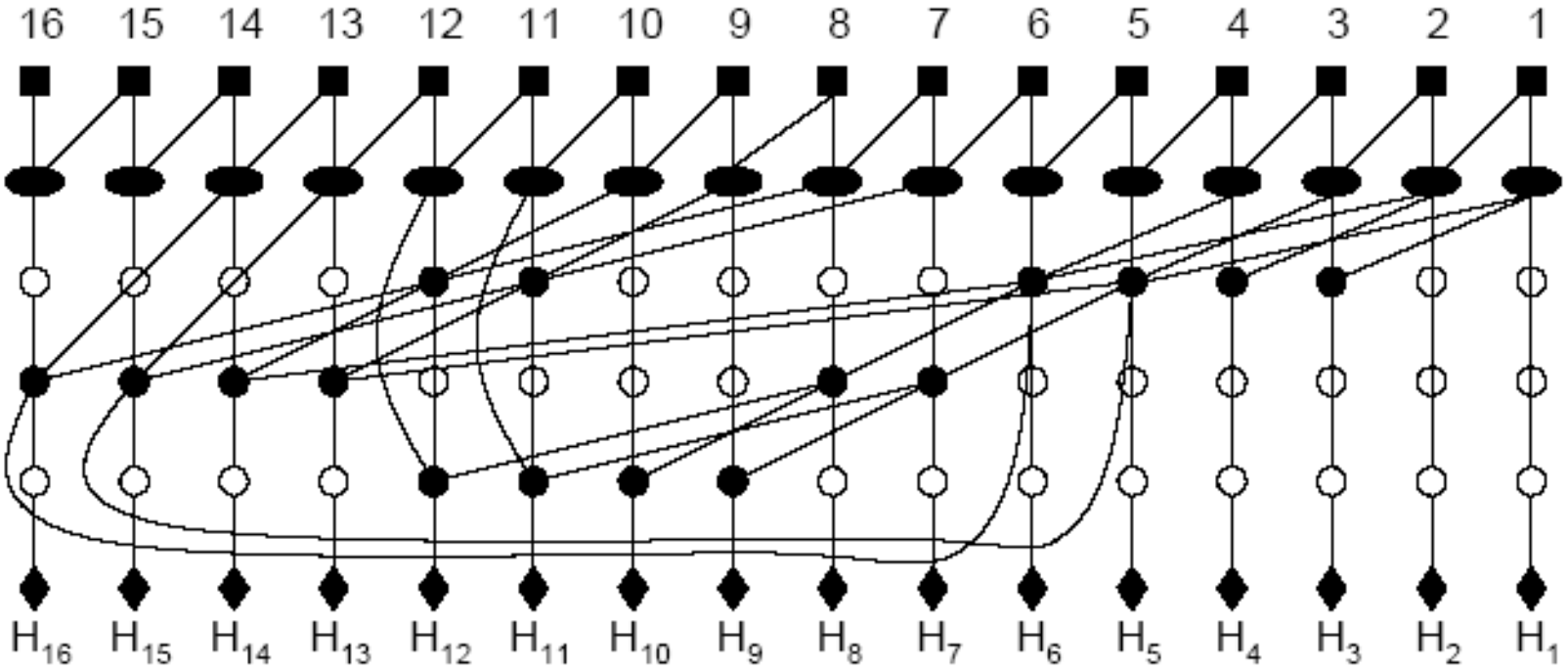
Increasing Required Time
(delay=34.5, power = 30.5FO4)



Decreasing Required Time
(delay=36.5, power = 32.5FO4)



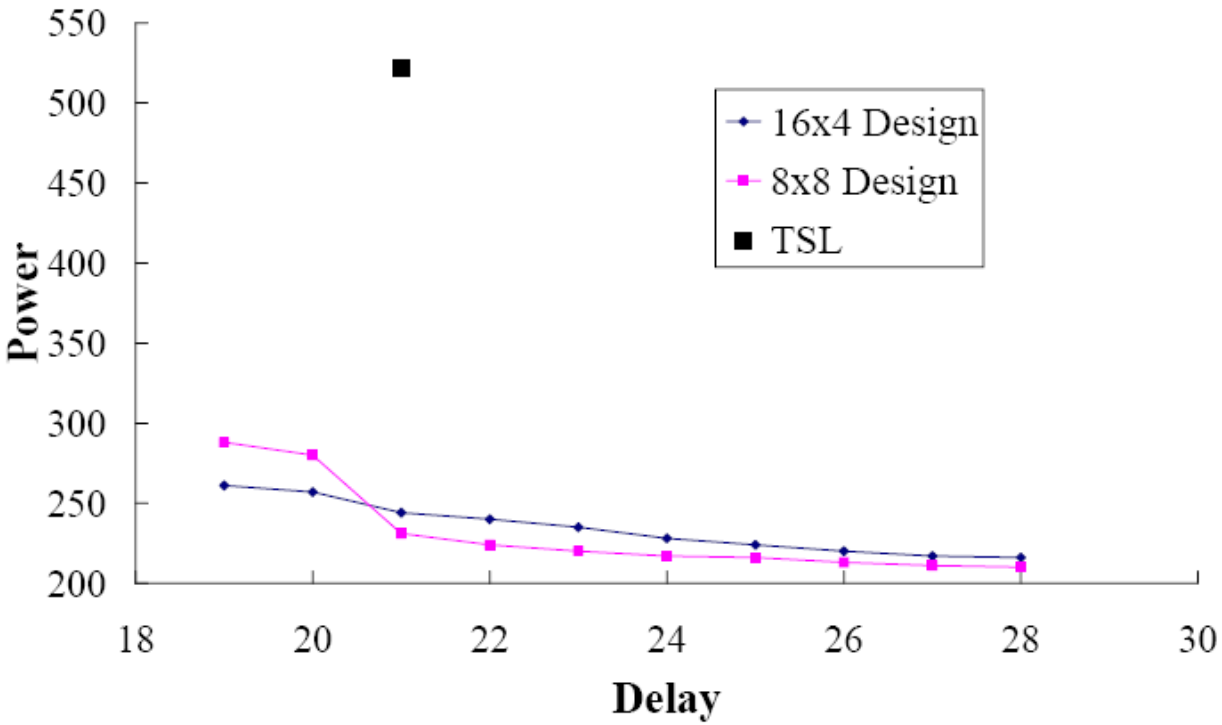
Convex Required Time
(delay=36.5, power = 32.5FO4)



Experiments – 64-bit Hierarchical Structure (Mixed-Radix)

- Handle high bit-width applications
- 16x4 and 8x8

Experiments – 64-bit Hierarchical Structure



TSL: a 64-bit high-radix three-stage Ling adder

Oklobdzija and B. Zeydel, "Energy-Delay Characteristics of CMOS Adders", *High-Performance Energy-Efficient Microprocessor Design,* pp. 147-170, 2006

ASIC Implementation - Results

- 64-bit hierarchical design (mixed-radix) by ILP vs. fast carry look-ahead adder by Synopsys Design Compiler
- TSMC 90nm standard cell library was

Method	Area(nm^2)	Delay (ns)	Power (mW)
DC	4473 (1)	1.1656 (1)	2.74 (1)
ILP	3833 (0.86)	0.9425 (0.81)	2.54 (0.93)
ILP	3636 (0.81)	0.9607 (0.82)	2.35 (0.86)
ILP	3114 (0.70)	1.1278 (0.97)	1.97 (0.72)

Future Work

- ILP formulation improvement
  - Expected to handle 32 or 64 bit applications without hierarchical scheme
- Optimizing other computer arithmetic modules
  - Comparator, Multiplier

Q & A

Thank You!