Optimum Prefix Adders in a Comprehensive Area, Timing and Power Design Space

Jianhua Liu\textsuperscript{1}, Yi Zhu\textsuperscript{1}, Haikun Zhu\textsuperscript{1}, John Lillis\textsuperscript{2}, Chung-Kuan Cheng\textsuperscript{1}

\textsuperscript{1}Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093

\textsuperscript{2}Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607
Outline

- Previous Works and Motivation
- Area/Timing/Power Model
- ILP Formulation of Prefix Adder
- Experimental Results
- Conclusions
Previous Works and Motivation

- Parallel prefix adder is the most flexible and widely-used binary adder for ASIC designs.
- Prefix network formulation:

  Pre-processing:
  \[ g_i = a_i b_i \]
  \[ p_i = a_i \oplus b_i \]

  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:
  \[ c_{i+1} = G_{[i:0]} + P_{[i:0]} \cdot c_0 \]
  \[ s_i = p_i \oplus c_i \]
Each output network is an alphabetical tree:

- Output i is the root of a binary tree covering inputs 1-i.
- An in-depth traversal of the tree terminals follows the sequence of the inputs.

**Previous Works and Motivation**
Previous Works and Motivation

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

*Max Fanouts is based on the regular buffer insertions at all empty space*
The design space of prefix adder is considered as the tradeoff among logical levels, max fanouts and wire tracks. *Harris D, "A Taxonomy of Parallel Prefix Networks" Nov. 2003.

Logical levels: \( L = \log_2 n + l \)
Max fanouts : \( F = 2^f + 1 \)
Wire tracks : \( T = 2^t \)
\[ l + f + t = \log_2 n - 1 \]

Timing: \( L \times F \times Dgp \)
\( (Dgp: \text{delay of one GP adder with unit load}) \)

Area: \( L \times (Hgp + T \times Hwt) \times n \)
\( (Hgp: \text{Height of one GP adder} \)
\( Hwt: \text{Height of one wire track} \)

Favor the minimal logical levels
Previous Works and Motivation

- Increasing impact of physical design.
- Power becomes a critical concern.
Previous Works and Motivation

- **Input:** bit width, physical area, input arrival times, output required times.
- **Output:** placed prefix adder
- **Constraint:** alphabetical tree rooted at each output $i$ to cover inputs $1$ to $i$, area and timing requirements
- **Objective:** minimize power consumption
Models – Area Model

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

\[ A = n \times m \]

- \( n \): Bit width
- \( m \): Physical depth
Models – Timing Model

- Cload includes both gate and wire capacitance. Wire capacitance is proportional to wire length.

\[ C_{load} = C_{wire} + C_{gate} \]
\[ C_{wire} = \lambda^w \times (H_{bb} + W_{bb}) \]
\[ C_{gate} = \sum C_{in} \]

\((\lambda^w = 0.5)\)

- Use a linear timing model derived from logical effort.

\[ \text{Delay}_{GP^l} = 1.5 \ C_{load} + 2.5 \]
\[ \text{Delay}_{GP^r} = 2.0 \ C_{load} + 2.5 \]

*Harris D, Sutherland I, ”Logical Effort of Carry Propagate Adders”, 2004.*
Models – Power Model

- Total power consumption:
  Dynamic power + Static Power

- Static power: leakage current of device
  \[ P_{sta} = \lambda^s \]  \( (\lambda^s = 0.5) \)

- Dynamic power: current switching capacitance
  \[ P_{dyn} = \rho \times C_{load} \]

- \( \rho \) is the switching probability
  \[ \rho = j \]  \( (j \) is the logical level*)

\[ P_{total} = P_{dyn} + P_{sta} = j \cdot C_{load} + \lambda^s \]

* Vanichayobon S, etc, “Power-speed Trade-off in Parallel Prefix Circuits”, 2002
ILP on Prefix Adder – Overall Picture

- We propose to formulate prefix computation as Integer Linear Programming (ILP) problem.
- Optimum solution can be produced by contemporary ILP solver.

Structure variables:
- GP adders
- Connections
- Physical positions

Capacitance variables:
- Gate cap
- Vertical wire cap
- Horizontal wire cap

Timing variables:
- Input arrival time
- Output arrival time

**Power objective**

Structure variables defines the ILP solution space
ILP – Linear Programming

- Linear Programming: linear constraints, linear objective, fractional variables.

Maximize:
\[ x_1 + 2x_2 + 3x_3 \]
subject to:
\[ -x_1 + x_2 + x_3 \leq 20 \]
\[ x_1 - 3x_2 \leq 30 \]

\[
\begin{bmatrix}
-1 & 1 & 1 \\
1 & -3 & 0 \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
\end{bmatrix}
\leq
\begin{bmatrix}
20 \\
30 \\
\end{bmatrix}
\]

*LP problems are polynomial time solvable (interior point algorithm, Karmarkar 1984)*
ILP – Integer Linear Programming

- Integer Linear Programming: all variables are integers.

*ILP problem with bounded variables is NP-hard.*
ILP – Branch and Bound

- Brach and bound with linear relaxation algorithm in ILP solvers:

Minimize $F(b_1, b_2, b_3, b_4, f_1, \ldots)$

$b_i$ is binary

Root (all vars are fractional)

Cut

Bound (Smallest candidate)

It is VERY helpful if ILP objective is close to LP objective
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: $x_1 \geq 300$
$\quad x_2 \geq 500$
$\quad x_3 = \min(x_1, x_2)$

**ILP formulation:**

Minimize: $x_3$
Subject to: $x_1 \geq 300$
$\quad x_2 \geq 500$
$\quad x_3 \leq x_1$
$\quad x_3 \leq x_2$
$\quad x_3 \geq x_1 - 1000 b_1$ \hspace{1cm} (1)
$\quad x_3 \geq x_2 - 1000 (1 - b_1)$ \hspace{1cm} (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 – Summary

- Integer Linear Programming is a powerful solution space search tool guided by Linear Programming.
- However, pseudo-linear constraints may compromise the efficiency.
ILP on Prefix Adder – Structure

- ILP decision variables represent GP adders and interconnects in logical view.
- Alphabetical tree rooted at each output.
  - Each GP adder has exact one left input and one right input. (*fanin const.*)
  - At least one input is from the previous level. (*logical level const.*)
  - Every GP adder roots an alphabetical tree covering a continuous segment. (*root const.*)
ILP on Prefix Adder – Structure

Variables:

- $gp(i,j) \{0,1\}$: GP adders in the $n \times d$ array ($d$: logical depth)
- $wl(i,j,h) \{0,1\}$: The wire from $(i,h)$ to the left fanin of $(i,j)$
- $wr(i,j,k,l) \{0,1\}$: The wire from $(k,l)$ to the right fanin of $(i,j)$

Constraints:

- **(fanin const.)** One left/right fanin for each GP adder
  \[
  \sum_h wl(i,j,h) = gp(i,j) \quad \forall (i,j) \quad i > h
  \]
  \[
  \sum_{(k,l)} wr(i,j,k,l) = gp(i,j) \quad \forall (i,j) \quad i > k \& j > l
  \]

- **(logical level const.)** At least one fanin from the previous level
  \[
  wl(i,j,j-1) + \sum_k wr(i,j,k,j-1) \geq gp(i,j) \quad \forall (i,j)
  \]
ILP on Prefix Adder – Structure

- The segment information is necessary for root constraint. The GP segments of two children must be adjacent.

**Variables:**
- \( gpl(i,j), gpr(i,j) \) int [1,n]:
  The segment covered by \( gp(i,j) \) is \([gpl(i,j):gpr(i,j)]\)

**Constraints:**
- (root const.) The GP segments of two children must be adjacent
  \[ gpl(i, j) = gpl(i, h) \text{ if } wl(i, j, h) = 1 \]  \( (1) \)
  \[ gpr(i, j) = gpr(k, l) \text{ if } wr(i, j, k, l) = 1 \]
  \[ gpr(i, h) = gpl(k, l) + 1 \text{ if } wl(i, j, h) = 1 \& wr(i, j, k, l) = 1 \]

**Conditional constraint (1) in ILP formulation:**
- \( gpl(i, j) \geq gpl(i, h) - n \cdot (1 - wl(i, j, h)) \)
- \( gpl(i, j) \leq gpl(i, h) + n \cdot (1 - wl(i, j, h)) \)
ILP on Prefix Adder – Structure

- Physical position variable attached to each GP adder describes physical level.
- No overlap in the physical view. *(overlap const.)*

**Variables:**
- $phy(i,j)$ int $[1,m]$: The physical position of $gp(i,j)$ is $(i, phy(i,j))$.  
  ($m$: physical depth)

**Constraints:**
- *(overlap const.)* Each physical position contains at most one GP adder

$$phy(i, j) \neq phy(i, h) \quad \forall i, j \neq h$$
ILP on Prefix Adder – Example

\[ gp(2,1)=1, \; wl(2,1,0)=1, \; wr(2,1,1,0)=1 \]
\[ gp(3,2)=1, \; wl(3,2,0)=1, \; wr(3,2,2,1)=1 \]
\[ gp(4,3)=1, \; wl(4,3,0)=1, \; wr(4,3,3,2)=1 \]

Logical view

\([gpl(2,1):gpr(2,1)] = [2:1]\)
\([gpl(3,2):gpr(3,2)] = [3:1]\)
\([gpl(4,3):gpr(4:3)] = [4:1]\)

Physical view

\[ phy(2,1)=1 \]
\[ phy(3,2)=1 \]
\[ phy(4,3)=2 \]
ILP on Prefix Adder – Capacitance

- Gate capacitance is calculated based on logical fanouts.
  - Gate cap equals to the number of fanouts, when input cap of GP adder is 1 unit. (*gate const.*)

- Wire capacitance depends on physical placement.
  - Vertical wire cap is proportional to the max vertical height of each fanout. (*wire const.*)
  - Horizontal wire cap is proportional to the max horizontal width of each fanout. (*wire const.*)
ILP on Prefix Adder – Capacitance

Variables:

- \( C_g(i,j) \) float: Gate load capacitance of \((i,j)\)
- \( C_wv(i,j) \) float: Vertical wire load capacitance of \((i,j)\)
- \( C_wh(i,j) \) float: Horizontal wire load capacitance of \((i,j)\)

Constraints:

- \((\text{gate const.})\) Gate load capacitance:
  \[
  C_g(i, j) = \sum_{h} w_l(i, h, j) + \sum_{(k,l)} w_r(k, l, i, j)
  \]
- \((\text{wire const.})\) Wire load capacitances:
  \[
  C_wv(i, j) \geq \lambda^w (\text{phy}(i, h) - \text{phy}(i, j)) \quad \text{if} \quad w_l(i, h, j) = 1 \quad (\lambda^w = 0.5)
  \]
  \[
  C_wv(i, j) \geq \lambda^w (\text{phy}(k, l) - \text{phy}(i, j)) \quad \text{if} \quad w_r(k, l, i, j) = 1
  \]
  \[
  C_wh(i, j) \geq \lambda^w (k - i) \quad \text{if} \quad w_r(k, l, i, j) = 1
  \]
ILP on Prefix Adder – Timing

- The output time is the max path delay. (*output const.*)
- Input arrival times equal to the output times of two children. (*input const.*)
- According to the timing model, gate delay is calculated based on load capacitance.

\[
\text{Delay}_{\text{GP}} = 1.5 \ C_{\text{load}} + 2.5 \\
\text{Delay}_{\text{GP'}} = 2.0 \ C_{\text{load}} + 2.5
\]
ILP on Prefix Adder – Timing

Variables:

• $T_l(i,j)$ float: Left input arrival time of $(i,j)$

• $T_r(i,j)$ float: Right input arrival time of $(i,j)$

• $T(i,j)$ float $[0, T_{max}]$: Output time of $(i,j)$

(T$_{max}$: Output required time.)

Constraints:

• (input const.) Input arrival times:

  $T_l(i,j) = T(i,h)$ if $wl(i,j,h) = 1$

  $T_r(i,j) = T(k,l)$ if $wr(i,j,k,l) = 1$

• (output const.) Output time:

  $T(i,j) \geq T_l(i,j) + 1.5 \cdot C_{load}(i,j) + 2.5$

  $T(i,j) \geq T_r(i,j) + 2.0 \cdot C_{load}(i,j) + 2.5$

  ($C_{load}(i,j) = C_g(i,j) + C_{wv}(i,j) + C_{wh}(i,j)$)
ILP on Prefix Adder – Power

- Total power consumption is the summation of power consumption on each GP adder.
- The objective is to minimize total power consumption.

\[
\text{Minimize : } \sum_{(i,j)} j \cdot C_{\text{load}}(i,j) + \lambda^S \cdot g_p(i,j)
\]
**ILP on Prefix Adder – Example**

C\(_{g}(2, 1)\) = 1, C\(_{wv}(2, 1)\) = 0, C\(_{wh}(2, 1)\) = 0.5
C\(_{g}(3, 2)\) = 1, C\(_{wv}(3, 2)\) = 0.5, C\(_{wh}(3, 2)\) = 0.5
C\(_{g}(4, 3)\) = 0, C\(_{wv}(4, 3)\) = 0, C\(_{wh}(4, 3)\) = 0

\(C_{load}(2, 1) = 1.5\)
\(C_{load}(3, 2) = 2\)
\(C_{load}(4, 3) = 0\)

\(T_{l}(2, 1) = 0, T_{r}(2, 1) = 0, T(2, 1) = 0 + 2 \times 1.5 + 2.5 = 5.5\)
\(T_{l}(3, 2) = 0, T_{r}(3, 2) = 5.5, T(3, 2) = 5.5 + 2 \times 2 + 2.5 = 12\)
\(T_{l}(4, 3) = 0, T_{r}(4, 3) = 12, T(4, 3) = 12 + 2 \times 0 + 2.5 = 14.5\)

Power = \(1 \times C_{load}(2, 1) + 2 \times C_{load}(3, 2) + 3 \times C_{load}(4, 3) + 3 \times 3 = 14.5\)
ILP on Prefix Adder – Extension

- Gate sizing and buffer insertion are two important optimization technologies to improve performance.
- Gate sizing: decrease gate delay, increase input capacitance.
- Buffer insertion: introduce new element, impact placement.
- Gate sizing and buffer insertion can be supported by ILP formulation.
Experimental Results

- Optimum prefix adders solved by CPLEX 9.1
- 8-bit prefix adders
  - Uniform input arrival time
  - Non-uniform input arrival time
- Hierarchical 64-bit prefix adders
- 64-bit prefix adder implementation (Synopsys flow, TSMC 90nm technology)
  - Module Compiler
  - Astro
  - Prime Power
## Experimental Results – 8-bit Uniform

<table>
<thead>
<tr>
<th>Method</th>
<th>Timing ($D_{FO4}$)</th>
<th>Depth</th>
<th>Power ($P_{FO4}$)</th>
<th>CPU (s)</th>
<th>Method</th>
<th>Timing ($D_{FO4}$)</th>
<th>Depth</th>
<th>Power ($P_{FO4}$)</th>
<th>CPU (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILP</td>
<td>10.0</td>
<td>1</td>
<td>20.1</td>
<td>0.31</td>
<td>K-S</td>
<td>6.2</td>
<td>3</td>
<td>29.0</td>
<td>-</td>
</tr>
<tr>
<td>ILP</td>
<td>10.0</td>
<td>2</td>
<td>17.5</td>
<td>124</td>
<td>ILP</td>
<td>6.0</td>
<td>2</td>
<td>20.9</td>
<td>259</td>
</tr>
<tr>
<td>ILP (S)</td>
<td>9.0</td>
<td>1</td>
<td>25.6</td>
<td>2.83</td>
<td>ILP</td>
<td>5.6</td>
<td>2</td>
<td>22.9</td>
<td>45.7</td>
</tr>
<tr>
<td>ILP</td>
<td>9.0</td>
<td>2</td>
<td>17.5</td>
<td>83.4</td>
<td>ILP (S)</td>
<td>5.6</td>
<td>2</td>
<td>21.6</td>
<td>756</td>
</tr>
<tr>
<td>ILP (S)</td>
<td>8.6</td>
<td>1</td>
<td>27.6</td>
<td>1.28</td>
<td>ILP</td>
<td>5.6</td>
<td>3</td>
<td>21.9</td>
<td>1237</td>
</tr>
<tr>
<td>ILP</td>
<td>8.6</td>
<td>2</td>
<td>17.5</td>
<td>93.2</td>
<td>ILP (S)</td>
<td>5.0</td>
<td>2</td>
<td>23.6</td>
<td>1208</td>
</tr>
<tr>
<td><strong>B-K</strong></td>
<td><strong>7.8</strong></td>
<td><strong>3</strong></td>
<td><strong>19.9</strong></td>
<td>-</td>
<td>ILP</td>
<td>5.0</td>
<td>3</td>
<td>25.6</td>
<td>4563</td>
</tr>
<tr>
<td>ILP</td>
<td>7.6</td>
<td>2</td>
<td>18.0</td>
<td>112</td>
<td>ILP</td>
<td>4.6</td>
<td>3</td>
<td>26.1</td>
<td>7439</td>
</tr>
<tr>
<td>ILP</td>
<td>7.0</td>
<td>2</td>
<td>18.6</td>
<td>99.6</td>
<td>ILP (S)</td>
<td>4.2</td>
<td>3</td>
<td>27.9</td>
<td>9654</td>
</tr>
<tr>
<td><strong>Skl</strong></td>
<td><strong>6.8</strong></td>
<td><strong>3</strong></td>
<td><strong>20.8</strong></td>
<td>-</td>
<td>ILP (S)</td>
<td>4.0</td>
<td>4</td>
<td>36.4</td>
<td>20211</td>
</tr>
</tbody>
</table>

*(S): Gate sizing, B-K: Brent-Kung, Skl: Sklansky, K-S: Kogge-Stone*
Experimental Results – 8-bit Uniform
Experimental Results – 8-bit Uniform

- Some typical ILP results:

  All the 8-bit fastest prefix adders have 4 logical levels
Experimental Results – 8-bit Non-Uniform

<table>
<thead>
<tr>
<th>Case</th>
<th>Power</th>
<th>Depth</th>
<th>Power*</th>
<th>Depth*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increasing arrival times</td>
<td>20.8</td>
<td>3</td>
<td>26.1</td>
<td>3</td>
</tr>
<tr>
<td>Decreasing arrival times</td>
<td>25.1</td>
<td>3</td>
<td>26.1</td>
<td>3</td>
</tr>
<tr>
<td>Convex arrival times</td>
<td>21.6</td>
<td>2</td>
<td>23.6</td>
<td>3</td>
</tr>
</tbody>
</table>

* Use the worst input arrival time for all inputs
Experimental Results – 64-bit Hierarchical

For high bit-width application, ILP method can be applied in a hierarchical design strategy.
Experimental Results – 64-bit Hierarchical

- Hierarchical ILP designs in solution space:
  (The physical depth is set to 6)

<table>
<thead>
<tr>
<th>Method</th>
<th>Timing ($D_{FO4}$)</th>
<th>Power ($P_{FO4}$)</th>
<th>Method</th>
<th>Timing ($D_{FO4}$)</th>
<th>Power ($P_{FO4}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical ILP</td>
<td>28</td>
<td>369</td>
<td>Hierarchical ILP</td>
<td>18</td>
<td>386</td>
</tr>
<tr>
<td><em>Brent-Kung</em></td>
<td>27</td>
<td>473</td>
<td>Sklansky</td>
<td>17</td>
<td>492</td>
</tr>
<tr>
<td>Hierarchical ILP</td>
<td>26</td>
<td>370</td>
<td>Hierarchical ILP</td>
<td>16</td>
<td>402</td>
</tr>
<tr>
<td>Hierarchical ILP</td>
<td>24</td>
<td>373</td>
<td>Hierarchical ILP</td>
<td>15</td>
<td>416</td>
</tr>
<tr>
<td>Hierarchical ILP</td>
<td>22</td>
<td>375</td>
<td><em>Kogge-Stone</em></td>
<td>15</td>
<td>3032</td>
</tr>
<tr>
<td>Hierarchical ILP</td>
<td>20</td>
<td>379</td>
<td>Hierarchical ILP</td>
<td>14</td>
<td>473</td>
</tr>
</tbody>
</table>
37

The power of Kogge-Stone add is much larger than other prefix adders.

Experimental Results – 64-bit Hierarchical
Experimental Results – 64-bit Hierarchical

- The fastest 64-bit hierarchical ILP adder:
Experimental Results – 64-bit Implementation

- 64-bit ILP prefix adders compared with 64-bit fast prefix adders generated by Module Compiler with relative placement.

<table>
<thead>
<tr>
<th></th>
<th>ILP</th>
<th>Module Compiler</th>
<th>Power Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Timing (ns)</td>
<td>Total Power [Wire Power] (mW)</td>
<td>Timing (ns)</td>
</tr>
<tr>
<td>0.74</td>
<td>1.9 [0.93]</td>
<td></td>
<td>0.75</td>
</tr>
<tr>
<td>0.76</td>
<td>1.8 [0.90]</td>
<td></td>
<td>0.83</td>
</tr>
<tr>
<td>1.13</td>
<td>1.15 [0.65]</td>
<td></td>
<td>1.24</td>
</tr>
</tbody>
</table>
Experimental Results – 64-bit Implementation

64-bit ILP Prefix Adder Physical View

64-bit MC Prefix Adder Physical View
Conclusions

- We propose an ILP method to solve minimal power prefix adders.
- The comprehensive area/timing/power model involves physical placement, gate/wire capacitance and static/dynamic power consumption.
- The ILP method can handle gate sizing, buffer insertion for both uniform and non-uniform input arrival time applications.
- The ILP method can be applied in hierarchical design methodology for high bit-width applications.
Thank You