Timing-Power Optimization for Mixed-Radix Ling Adders by Integer Linear Programming

Yi Zhu, Jianhua Liu, Haikun Zhu and Chung-Kuan Cheng
Department of Computer Science, University of California, San Diego
La Jolla, CA, 92093-0404, U.S.A.
{y2zhu,jhliu,hazhu,ckcheng}@ucsd.edu

Abstract— This paper optimizes timing and power consumption of mixed-radix Ling adders with the physical area constraints using an integer linear programming formulation. Each cell in the prefix network is flexible to have different radix and size, and Ling carries are incorporated. Optimal solutions are obtained by solving the proposed formulation. The experiments show that the produced optimal structures have a large power saving compared with traditional designs. The ASIC implementation results are superior to those produced by Synopsys Module Compiler.

I. Introduction

As the most fundamental and commonly used operation in computer arithmetic, binary addition has been extensively studied in the data path research. There have been a large variety of algorithms to construct adder structures, among which parallel prefix adders are popular since they have simple cells and offer high flexibility. Three classic regular structures, Sklansky [12], Kogge-Stone [7] and Brent-Kung [1] adders, can achieve minimal logic levels, wire tracks, or fanouts. In addition to these regular structures, Zimmermann proposed a non-heuristic algorithm to produce irregular prefix networks that are able to deal with non-uniform input arrival times [15]. In [14] zero-deficiency adder was defined, which can reach the lower bound of number of components for given number of logic levels. In [9] Liu et al. used integer linear programming to identify optimal prefix structures to minimize power consumption, and more practical area, timing and power models are adopted in their formulation.

However, as most of the previous works, the work in [9] is also restricted to the traditional radix-2 adders, and do not make use of Ling adder, which is a popular and efficient technique to speed up prefix adders [8]. Those who considered these factors, such as [6] and [4], only explored regular structures. In this paper, we propose an integer linear programming (ILP) formulation to optimize more generalized prefix Ling adders. The contributions of this paper include:

- We devise the ILP formulation so that the commercial ILP solver CPLEX is able to produce minimum power solutions with given structural, area and timing constraints. Hence it is a useful tool to generate candidates for back-end design. Furthermore, it is very easy to adjust our formulation with different parameters and constraints, such as regular radix constraints or the relative ratio of static and dynamic power consumption. It actually provides a flexible framework for designers to design customized adders. We also develop a program that can automatically synthesize adders according to the ILP results; therefore our approach can be directly used in Synopsys design flow.
- The formulation we devise is able to effectively depict the structure, timing and power properties of prefix adders using integer decision variables and linear constraints, which are the essential conditions for the ILP solver to seek for optimal solutions. Since prefix adders have a large design space, optimal solutions are usually hard to identify without good formulations. We work out several sets of constraints to prune the search space so that the running time is significantly reduced.
- Mixed-radix adders are formulated in our model, i.e. a GP cell could have different radix 2, 3, 4, and a prefix network can contain cells with different radix. This feature provides more flexibility to designers to construct fast adders. High-radix adders are popular for high performance applications such as microprocessor ALUs [10][11]. A high-radix adder requires less logic levels; however, a single high-radix cell has greater logical effort, parasitic delay and more power consumption. These factors are all taken care of in our ILP formulation.
- We construct prefix Ling adders, which are able to compute bit sum and carry signals faster by using a simplified form of carry lookahead equations [8]. The experiments show that Ling adders are able to produce better results than normal prefix adders.
- We apply hierarchical design methods to handle high bit-width applications. One weakness of ILP solver is the unscalable computational time. So we propose a divide-and-conquer strategy to optimize 64-bit adders in multiple blocks. We show that the results are superior to the 64-bit adders produced by Synopsys Module Compiler.

The rest of the paper is organized as follows. The prefix structure, Ling adders and mixed-radix adders are introduced in the next section. Their area, timing and power models, are discussed in Section III. The ILP formulation is described in Section IV; as the entire formulation is complex, we shall present how to formulate the properties of Ling and mixed-radix adders in more details. Several categories of experimental results, including the ASIC implementation, are presented in Section V. Conclusions are given in the last section.

II. Preliminaries

A. Parallel Prefix Adders

Assuming $A = a_n \dots a_2 a_1$ and $B = b_n \dots b_2 b_1$ are two operands in the addition, with 1-bit carry-in c_0, a prefix adder can be considered as a three-stage circuit. In the preprocessing step, the *generate* bit g_i and the *propagate* bit p_i are produced for each input i, where $g_i = a_i b_i$, and $p_i = a_i \oplus b_i$. In the second stage, the generate and propagate computation is extended to multiple bits. They are defined as:

$$G_{[i:j]} = \begin{cases} g_i & \text{if } i = j \\ G_{[i:k]} + P_{[i:k]} G_{[k-1:j]} & \text{if } i \geq k > j \end{cases} \qquad (1)$$

$$P_{[i:j]} = \begin{cases} p_i & \text{if } i = j \\ P_{[i:k]} P_{[k-1:j]} & \text{if } i \geq k > j \end{cases} \quad (2)$$

Finally, in the third stage, the sum s_i and carry c_i can be calculated from G and P as $s_i = p_i \oplus c_{i-1}$ and $c_i = G_{[i:0]}$.

We use the prefix operator $\bullet$ to simplify the notation of (G, P) computation:

$$(G, P)_{[i:j]} = (G, P)_{[i:k]} \bullet (G, P)_{[k-1:j]} \quad (3)$$

Interpret the prefix operator $\bullet$ as a node and the signal (G, P) as an edge in a graph, the prefix computation structures can be viewed as directed acyclic graphs. Fig. 1 shows an 8-bit Brent-Kung adder. The symbols ■, ● and ▼ represent gp generators, GP cells and sum generators respectively.

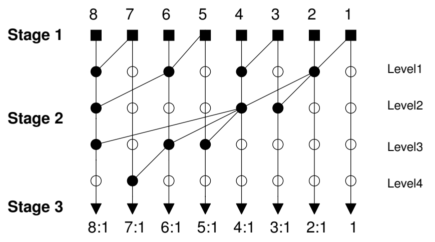


Fig. 1. The 8-bit Brent-Kung Prefix Adder

B. Ling Adders

In our formulation and implementation, we follow the approach in [4] that presented a prefix formulation which takes advantage of the simplicity of Ling equations. Ling adders are a variation among the commonly used carry lookahead adders. Consider the adjacent bit pairs (a_i, b_i) and (a_{i-1}, b_{i-1}), Ling defined the i^{th} Ling pseudo carry as $H_i = c_i + c_{i-1}$ [8]. Therefore, H_i can be expressed by the g and p signals as

$$H_i = g_i + g_{i-1} + p_{i-1} \cdot g_{i-2} + \ldots + p_{i-1} \cdot p_{i-2} \cdot \ldots \cdot p_1 \cdot g_0 \quad (4)$$

The main advantage of Ling adders is that H_i can be computed faster than the corresponding carry c_i since it is derived from a simpler Boolean function. However, the derivation of final sum bits is a little more complicated. As we know that $c_i = p_i \cdot H_i$, it holds that $s_i = (a_i \oplus b_i) \oplus c_{i-1} = (a_i \oplus b_i) \oplus (p_{i-1} \cdot H_{i-1})$. It can be further transformed as follows:

$$s_i = \overline{H_{i-1}} \cdot (a_i \oplus b_i) + H_{i-1} \cdot ((a_i \oplus b_i) \oplus p_{i-1}) \quad (5)$$

To obtain the prefix formulation of Ling adders, first the g and p signals of adjacent bits are combined as

$$G_i^* = g_i + g_{i-1} \quad \text{and} \quad P_i^* = p_i \cdot p_{i-1} \quad (6)$$

Then the group signals $(G^*_{[i:j]}, P^*_{[i:j]})$ is defined as follows

$$(G^*_{[i:j]}, P^*_{[i:j]}) = (G_i^*, P_{i-1}^*) \bullet (G_{i-2}^*, P_{i-3}^*) \bullet \ldots \bullet (G_j^*, P_{j-1}^*) \quad (7)$$

The meaning of the prefix operator $\bullet$ remains the same. Note that there is one offset in the index of a GP pair, and the index offset between consecutive pairs are two. The recursive formulation can be written as

$$(G^*_{[i:j]}, P^*_{[i:j]}) = (G^*_{[i:k]}, P^*_{[i:k]}) \bullet (G^*_{[k-2:j]}, P^*_{[k-2:j]}) \quad (8)$$

The Ling carry H_i is equal to $G^*_{[i:1]}$. Compared with the recursive formulation (3), the lower part signal is from $k-2$ instead of $k-1$, which indicates that we can separate odd and even bits: odd bits signals are always from lower odd bits and even bits signals are always from lower even bits.

An example of Lander-Fischer prefix Ling adder is shown in Fig. 2. Note that in the first level of the prefix network, there are ellipses instead of circles, which means the cells are simpler than normal GP cells, since the G_i^* is computed by only g signals and merely an OR operation is needed.

This simplification mainly affects the performance in three aspects: first, we could obtain faster g signals than normal prefix adders; second, lower bit P signals are used in each bit — note in Equation (7), the pair $(G^*_{[i:j]}, P^*_{[i:j]})$ is generated by P^*_{i-1} instead of P^*_i; finally, the simpler cells in the first level also consume less power. As the trade-off, the sum generator is more complicated and requires an XOR gate and a multiplexer instead of a single XOR gate in the traditional design, as shown in Fig. 2.

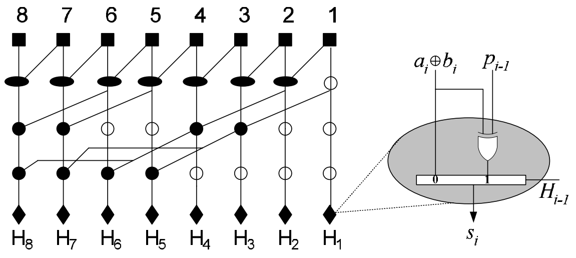


Fig. 2. The 8-bit Lander-Fischer Ling Adder

C. Mixed-Radix Adders

In mixed-radix adders, each GP cell in the prefix adder may have number of fanins larger than 2. They need less logic levels but each GP cell is more complicated and has larger delay and power consumption. In this work we consider GP cells may have radix 2, 3 or 4. Consider the flexibility, we allow different radix GP cells to appear in the same prefix network — it is easy to revise the formulation to impose regularity. The prefix recursive formula should be revised to have multiple operands. For example, for a radix-4 GP cell, the formula (8) is revised as

$$\begin{aligned}(G^*_{[i:j]}, P^*_{[i:j]}) = (G^*_{[i:k_1]}, P^*_{[i:k_1]}) \bullet (G^*_{[k_1-2:k_2]}, P^*_{[k_1-2:k_2]}) \\ \bullet (G^*_{[k_2-2:k_3]}, P^*_{[k_2-2:k_3]}) \bullet (G^*_{[k_3-2:j]}, P^*_{[k_3-2:j]})\end{aligned} \quad (9)$$

The timing and power models for different radix GP cells will be discussed in the following section.

III. Models

A. Area Model

We assume that prefix adders will keep the bit-slice structure in the placement, while each column will be compactly placed so that there will not be empty positions between GP cells in each column. Fig. 3 illustrates the compact placement of the 8-bit Lander-Fischer Ling adder in Fig. 2. Note that, for a specific adder, the physical depth of the placement is always no more than its logical depth.

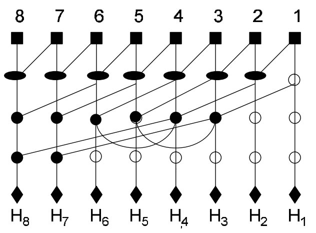
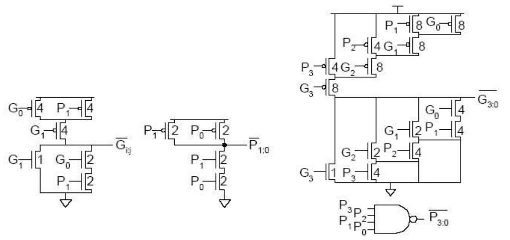


Fig. 3. Compact Placement for the 8-bit Lander-Fischer Ling Adder



(a) Radix-2 GP Cell Gates (b) Radix-4 GP Cell Gates

Fig. 4. Radix-2 and Radix-4 GP Cells

Radix	Term	Value	Radix	Term	Value
2	PDg	10/3	4	PDg	20/3
2	PDp	3	4	PDp	5
2	LEgL	5/3	4	LEgL	3
2	LEgR1	2	4	LEgR1	10/3
2	LEpL	10/3	4	LEgR2	10/3
2	LEpR1	4/3	4	LEgR3	4
3	PDg	16/3	4	LEPL	14/3
3	PDp	4	4	LEpR1	4
3	LEgL	7/3	4	LEpR2	6
3	LEgR1	8/3	4	LEpR3	2
3	LEgR2	3			
3	LEpL	10/3			
3	LEpR1	14/3			
3	LEpR2	5/3			

TABLE I
PARASITIC DELAYS & LOGICAL EFFORTS FOR HIGH RADIX GP CELLS

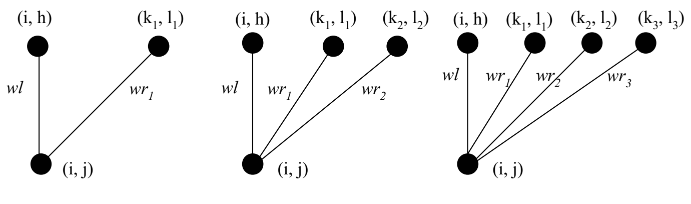
B. Timing Model

We use a linear timing model based on logical effort method to estimate the delay in our formulation. It has been shown that logical effort method is able to predict absolute delay within 5% – 20% of HSPICE [2][3]. We use the logical effort derivation similar to that in [6] and [9]. To incorporate the features of mixed-radix and Ling adders, we need to emphasize the following two enhancements.

First, we distinguish the timings of G and P signals for each GP cell. In the previous work [9], it is assumed that G path is always slower than the P path. In fact, this assumption is not always valid, especially when the input arrival times are non-uniform.

Second, we consider GP cells with mixed radix in our model. Hence different GP cell structures and logical effort values should be used to distinguish them. The structures for raidx-2 and radix-4 GP cells are shown in Fig. 4 [5]; the parasitic delay and logical effort values of various inputs for radix 2, 3 and 4 GP cells are listed in Table I.

In Table I, PD denotes the parasitic delays, LE denotes the input logical efforts, “g” and “p” indicate the signals for G and P, and left and right inputs are denoted by “L” and “R” respectively. For high-radix GP cells, the right inputs are ordered from left to right, as illustrated in Fig. 5 (The symbols in the brackets indicate the logic positions of the components). For example, LEgR2 represents the logical effort of G signal in the second right input.



(a) Radix-2 Adder (b) Radix-3 Adder (c) Radix-4 Adder

Fig. 5. Adder Input Structures

Given the logical effort and parasitic delay, the delay of a cell for G and P signals, denoted as T^G and T^P, can be calculated using the following formula respectively:

$$T^G(R) = \max\{\max_{i\in[1..R]}\{T_i^G + \textbf{LE\_G}(\text{R,i})\cdot\frac{\text{Output Load}}{\text{Input Cap}}\}, \max_{i\in[1..R-1]}\{T_i^P + \textbf{LE\_P}(\text{R,i})\cdot\frac{\text{Output Load}}{\text{Input Cap}}\}\} + \textbf{PD\_G}(R) \quad (10)$$

$$T^P(R) = \max_{i\in[1..R]}\{T_i^P + \textbf{LE\_P}(R,i)\cdot\frac{\text{Output Load}}{\text{Input Cap}}\} + \textbf{PD\_P}(R) \quad (11)$$

where R is the radix of the cell and can be 2, 3 or 4; T_i^G and T_i^P indicate the timing of the i^{th} input (from left to right) for G and P signals; and **LE_G**, **LE_P**, **PD_G** and **PD_P** denote the logical effort and parasitic delay for G and P signals. They are the functions of the cell radix R and input i, and can be looked up in Table I. The unit used to measure the delay is the parasitic delay of a single invertor, which is $\frac{1}{5}$ fanout-of-4 delay.

The output load consists of gate capacitance and wire capacitance. The ratio of the gate capacitance and input capacitance is the number of fanouts. To estimate the wire capacitance, we follow the method in [9] to calculate the total wire length of the bounding box around all the fanouts in the physical placement. A scaling factor 0.5 is applied to the total wire length, as suggested by [6].

When doing the gate sizing, we assume the input capacitance is inversely proportional to the size and the parasitic delay keeps constant; and the load capacitance will linearly increase if a cell drives a larger output, so we should use “normalized fanouts” instead of fanouts, which is the weighted sum of its fanouts with sizing. E.g. when a cell drives two cells with size 2 and 1 respectively, the normalized fanouts are 3. Consequently, we have the following equation that can be substituted to the delay calculation formula (10) and (11):

$$\frac{\text{Output Load}}{\text{Input Cap}} = \frac{\text{\#normalized fanouts} + 0.5\cdot\text{wire length}}{\text{size}} \quad (12)$$

In addition, we do not count the timing delay for the gates in the preprocessing stage (g, p signals generation) and postprocessing stage (sum calculation) in our formulation, since they keep unchanged for different prefix structures. Their power consumption is omitted as well.

C. Power Model

We consider both dynamic power and static power consumption. The dynamic power consumption is mainly due to the charging and discharging of capacitance and measured by the switching activities. We make use of the analysis in [13] about the switching activities in prefix adders and further consider the effect of load capacitance. Let d be the total number of logic levels and C_i be the total load capacitance in level i, the total dynamic power is

$$\sum_{i=1}^{d} i\cdot C_i \quad (13)$$

The static power contributes a significant portion to the total power consumption nowadays.We use the total number of GP cells to measure the total static power. In addition, two factors must be taken into account. First, the static power is increasing along with the size of a GP cell, here we assume this relation is linear; second, GP cells with different radix have different static power consumption. Consequently, the static power consumption of a single GP cell depends on both its size and radix, which has been considered in our ILP formulation. All the power consumption are measured by the unit $\frac{1}{4}$ FO4 switching power consumption.

IV. ILP Formulations

The ILP formulation is described in this section. Due to the space limitation, we shall only briefly introduce the main idea, and focus on how to formulate the unique constraints due to Ling adders and high-radix adders, as well as those constraints that can reduce the search space and running time.

A. Structural Constraints

Three types of constraints, named as **Bit-slice constraints**, **GP constraints** and **Overlap constraints**, were used to ensure the ILP solution has a valid structure in the $n \times d$ array, where n is the bit-width and d is the logical depth; and a series of variables were introduced to depict the prefix structure. The notations are expanded as follows:

- $x_{(i,j)} \in \{0,1\}$: 1 if and only if a GP cell is located in the i^{th} bit and j^{th} level (or say column i and row j); furthermore, $x_{(i,1)} = 1$ for all $i \in [1..n]$, since the first level of Ling adder is fixed.
- $w^L_{(i,j,h)} \in \{0,1\}$: 1 if and only if there is a left fanin wire to position (i,j) from position (i,h), and $h < j$.
- $w^{R1}_{(i,j,k,l)}, w^{R2}_{(i,j,k,l)}, w^{R3}_{(i,j,k,l)} \in \{0,1\}$: 1 if and only if there is a first (second, third) right fanin WR1 (WR2, WR3) (refer to Fig. 5) to position (i,j) from position (k,l), and $k < i, l < j$.
- $w^Z_{(i,j)} \in \{0,1\}$: 1 if and only if the output of a GP cell (i,j) connects to the primary output in column i.
- $y^L_{(i,j)}, y^R_{(i,j)} \in [1,n]$: the left and right bounds of a GP cell (i,j), which means the GP cell covers the range $[y^L_{(i,j)} : y^R_{(i,j)}]$.
- $p(i,j) \in [0,m]$: the physical level of a GP cell (i,j). The physical position will be $(i, p_{(i,j)})$.

The **Bit-slice constraints** ensure the bit-slice structure is preserved in each column: the left fanin is always from some one above in the same column, and the three right fanins are from the top-right quadrant. They can be formulated using the following constraints:

$$\sum_h w^L_{(i,j,h)} = x_{(i,j)} \qquad \forall i > h \tag{14}$$

$$\sum_{k,l} w^{R1}_{(i,j,k,l)} = x_{(i,j)} \quad \forall i > k \,\&\, j > l \tag{15}$$

$$\sum_{k,l} w^{R2}_{(i,j,k,l)} \le x_{(i,j)} \quad \forall i > k \,\&\, j > l \tag{16}$$

$$\sum_{k,l} w^{R3}_{(i,j,k,l)} \le x_{(i,j)} \quad \forall i > k \,\&\, j > l \tag{17}$$

$$\sum_j w^Z_{(i,j)} = 1 \qquad \forall i \tag{18}$$

Note that (17) and (18) use "$\le$" instead of "=", since the second and third right inputs are optional for radix-2 and radix-3 cells.

We impose the following two constraints to make sure wr_2 is always right to wr_1 and wr_3 is always right to wr_2. In fact they are not necessary to guarantee the feasibility since the order will be imposed by the later **GP constraints**. But it does help to prune the search space and speed up the optimization process.

$$\sum_l w^{R1}_{(i,j,k_1,l)} + \sum_l w^{R2}_{(i,j,k_2,l)} <= 1 \quad \text{if } k_1 \le k_2 \tag{19}$$

$$\sum_l w^{R2}_{(i,j,k_2,l)} + \sum_l w^{R3}_{(i,j,k_3,l)} <= 1 \quad \text{if } k_2 \le k_3 \tag{20}$$

The **GP constraints** are used to guarantee the obtained logical structure is a feasible prefix network, i.e. each GP cell covers a certain bit range determined by its inputs, which are ordered consecutive intervals, as expressed in Equation (9). The primary output at column i must cover the interval $[i, 1]$. They are implemented by calculating the values y^L and y^R. Since we are considering mixed-radix cells, 2, 3 or 4 intervals need to be checked. We are not enumerating the entire bunch of constraints here due to the space limitation. As examples, the left bound is equal to the left bound of its left input:

$$y^L_{(i,j)} = y^L_{(i,h)} \qquad \text{if } w^L_{(i,j,h)} = 1 \tag{21}$$

And the right bound of the left input is adjacent to the left bound of the first right input:

$$y^R_{(i,h)} = y^L_{(k,l)} + 1 \qquad \text{if } w^L_{(i,j,h)} = w^{R1}_{(i,j,k,l)} = 1 \tag{22}$$

The similar constraints exist to check the boundaries for other inputs, and additional constraints are needed to take care of the cases when WR2 and WR3 are absent. In [9], the above constraints must be transformed to "pseudo-linear" constraints, which do harm to the performance of the ILP solver. For instance, constraint (21) needs to be written as

$$y^L_{(i,j)} \ge y^L_{(i,h)} - n \cdot (1 - w^L_{(i,j,h)}) \tag{23}$$

$$y^L_{(i,j)} \le y^L_{(i,h)} + n \cdot (1 - w^L_{(i,j,h)}) \tag{24}$$

However, we observe that the left bound of a specific cell is always equal to its column index, thus constraint (21) can be simply written as a linear constraint:

$$y^L_{(i,j)} = i \tag{25}$$

and constraint (22) can be simplified to:

$$y^R_{(i,h)} = \sum_{(k,l)} k \cdot w^{R1}_{(i,j,k,l)} + 1 \qquad \text{if } w^L_{(i,j,h)} = 1 \tag{26}$$

Although this simplification cannot eliminate all the pseudo-linear constraints (constraints with respect to the right bound cannot be simplified in this way), it is able to reduce around half of them for **GP constraints** and significantly improve the performance.

The **Overlap constraints** are simply to guarantee that no two cells are placed on the same physical position when placing the prefix network on a physical $n \times m$ array, where m is the physical depth. The constraint is written as

$$p_{(i,j)} \neq p_{(i,h)} \qquad \forall j \neq h \tag{27}$$

B. Timing Constraints

We can compute the timing delay $T^G_{(i,j)}$ and $T^P_{(i,j)}$ for cell (i,j) by decomposing the Max operators in formula (10) and (11) into a series of inequalities which ensure the values are greater than all the expressions inside the Max operators. But there are two factors, the radix R and the sizing variables, exist in the formula. In order to ensure the constraints are linear, we must avoid formulating these two parameters as variables directly. An "incremental" method is used in our formulation to handle this issue, by taking the advantage that both the radix and sizing have very limited number of discrete values — radix can only be 2, 3 or 4, and we only allow cells to be enlarged to 2, 3 or 4 times of the original size. For example, the parasitic delay of G signal for cell (i,j) can be expressed as

$$\frac{10}{3} + 2 \cdot \sum_{(k,l)} w^{R2}_{(i,j,k,l)} + \frac{4}{3} \cdot \sum_{(k,l)} w^{R3}_{(i,j,k,l)} \tag{28}$$

where $\frac{10}{3}$ is the parasitic delay for a radix-2 cell, which is considered as the "base case"; 2 is the incremental value from radix-2 to radix-3 ($\frac{16}{3} - \frac{10}{3}$, referring to Table I); $\sum_{(k,l)} w^{R2}_{(i,j,k,l)}$ indicates whether the second right input exists, or say, whether it is radix-3 or radix-4; and $\frac{4}{3}$ is the incremental value from radix-3 to radix-4. We use binary variables $size^2_{(i,j)}$, $size^3_{(i,j)}$ and $size^4_{(i,j)}$ to indicate the size of cell (i,j). Consequently, these two factors can be taken care of by incremental constraints in the similar way as above.

Referring to the delay calculation formula (10) and (11) in Section III.B, to obtain a delay for a specific cell, we should compute its number of normalized fanouts and output wire length. The fanouts is calculated as

$$c^G_{(i,j)} = \sum_h w^L_{(i,h,j)} + \sum_{(k,l)} w^{R1}_{(k,l,i,j)} + \sum_{(k,l)} w^{R2}_{(k,l,i,j)} + \sum_{(k,l)} w^{R3}_{(k,l,i,j)} + w^Z_{(i,j)} \tag{29}$$

and it can be adjusted to normalized fanouts (fanouts with output load sizing) by using the incremental method mentioned above.

The wire length is estimated by the half perimeter of the bounding box covering all fanouts, which is the maximal vertical and horizontal distance of each fanout. Hence the following constraints can be used to compute vertical and horizontal wire length:

$$c^{WV}_{(i,j)} \geq p_{(i,h)} - p_{(i,j)} \qquad \text{if } w^L_{(i,h,j)} = 1 \tag{30}$$

$$c^{WV}_{(i,j)} \geq p_{(k,l)} - p_{(i,j)} \qquad \text{if } w^{R1}_{(k,l,i,j)} + w^{R2}_{(k,l,i,j)} + w^{R3}_{(k,l,i,j)} \geq 1 \tag{31}$$

$$c^{WH}_{(i,j)} \geq k - i \qquad \text{if } w^{R1}_{(k,l,i,j)} + w^{R2}_{(k,l,i,j)} + w^{R3}_{(k,l,i,j)} \geq 1 \tag{32}$$

All these three are pseudo-linear constraints. However, we note that $w^{R1}_{(k,l,i,j)} + w^{R2}_{(k,l,i,j)} + w^{R3}_{(k,l,i,j)} \leq 1$ for each tuple (i,j,k,l), therefore constraint (32) can be revised to a linear constraint:

$$c^{WH}_{(i,j)} \geq (k-i) \cdot (w^{R1}_{(k,l,i,j)} + w^{R2}_{(k,l,i,j)} + w^{R3}_{(k,l,i,j)}) \tag{33}$$

Hence the load portion for cell (i,j) in the delay calculation, as shown in Equation (12), denoted as $c_{(i,j)}$, is computed as

$$c_{(i,j)} = \frac{c^G_{(i,j)} + 0.5 \cdot c^{WV}_{(i,j)} + 0.5 \cdot c^{WH}_{(i,j)}}{size_{(i,j)}} \tag{34}$$

C. Power Consumption Objective

According to the discussion in Section III.C, the total power consumption includes the dynamic power and static power, which depend on the logic level, output load, and number of cells respectively and can be formulated as

$$\text{Minimize} \sum_{(i,j)} j \cdot (c^G_{(i,j)} + 0.5 \cdot c^{WV}_{(i,j)} + 0.5 \cdot c^{WH}_{(i,j)}) + \lambda \sum_{(i,j)} x_{(i,j)} \tag{35}$$

where λ is the scaling factor to adjust the ratio between dynamic and static power.

V. Experimental Results

We conducted four categories of experiments: first we explored the delay and power trade-off of 16-bit optimal adders when the uniform arrival/required time is assumed; then we demonstrated the flexibility of our formulation by conducting experiments with non-uniform arrival/required time; a hierarchical design methodology was then used to build 64-bit adders, which were implemented using Synopsys Datapath design flow and the results were compared with those generated by Synopsys Module Compiler. Throughout our experiments, the ILP formulation was solved by ILOG CPLEX 9.1 solver; in the ASIC implementation, the area, timing and power values were reported by Physical Compiler, Astro and Prime Power respectively.

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	-

TABLE II
16-BIT OPTIMAL LING ADDERS RESULTS

A. Uniform Input Arrival Time

To test the timing and power trade-off of different adder structures, we apply our ILP formulation to find optimal solutions for 16-bit adders with different timing delay constraints. In the experiments, we test three types of adders: traditional radix-2 adders, adders with radix-2 and radix-4 cells only (sometimes radix-3 cells are not preferred in design), and mixed-radix adders which allow cells to have radix 2, 3 or 4. For each type, we conduct a series of experiments with various delay constraints, from loose to tight. The results are shown in Table II and the power/delay tradeoff curves are plotted in Fig. 6. For comparison, we plot the curves using dashed lines that represents the results for normal prefix adders without Ling carries. In addition, in Table II and Fig. 6, we also show the delay and power consumption for three classical structures, Brent-Kung (B-K), Sklansky, and Kogge-Stone (K-S) adders with Ling carries . All the timing and power results are normalized to FO4 delay (D_{FO4}) and FO4 switching power (P_{FO4}). Note that for each curve, a certain timing interval is sufficient to show the tradeoff, since the power consumption won't be decreased if it is bounded by the adder structure itself but not the delay constraint; and there are no feasible solutions if the delay constraint is too tight.

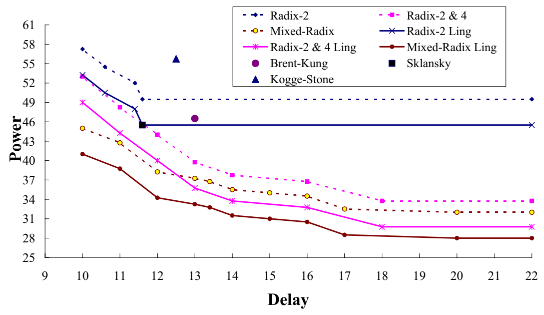


Fig. 6. Power Delay Trade-off for 16-bit Ling Adders

We have the following observations based on the results:

- Given the same delay constraint, radix-2&4 adders consumes less power than radix-2 adders and mixed-radix adders have even lower power consumption. This is understandable since the solution space for high radix-adders is larger; though each high-radix cell consumes more power, the entire network has less components as well as logic levels. For example, Figure 7, 8 and 9 show the minimum power 16-bit radix-2, radix-2&4 and mixed-radix Ling adders without any delay constraints respectively. We can observe that with radix-4 cells we can save logic levels, and with radix-3 cells we can further push the cells to lower logic levels. Therefore, the structure shown in Figure 8 saves 34.6% power consumption than that in Figure 7, and the adder in Figure 9 saves 38.5% power consumption than that in Figure 7. The mixed-radix adders show more advantages when compared with classic adders: with the same delay constraint, the mixed-radix adders can save 25.3%, 14.8% and 38.6% power consumption compared with Brent-Kung, Sklansky and Kogge-Stone adders respectively.

- The gap between traditional radix-2 adders and radix-2&4 adders are much larger with moderate delay constraints, which shows that high-radix has significant effect under this circumstance; radix-3 cells only have small marginal improvement and may not be preferred in real design. For example, if the delay constraint is 14 FO4 delay, the optimal radix-2&4 adder saves 25.8% power consumption than the radix-2 adder, and the the mixed-radix adder saves 30.7 %. For tight delay constraints, however, radix-3 cells are still beneficial. When the constraint is 10 FO4 delay, the radix-2&4 adder and mixed-radix adder save 8.0% and 23.0% power over the radix-2 adders, which indicates that mixed-radix adders are strongly preferred when tight delay constraints are imposed.
- The trade-off curves are not entirely concave, but a bit bumpy. It is because we are searching a discrete solution space. The adder structures, radix and sizing options, are all discrete variables. Hence the feasible solutions are isolated points in the entire space. However, we can still observe that power consumption is gradually increased when delay constraints become tighter and the increment is sharp when we demand for a very fast adder.
- Ling adders consume less power than normal prefix adders for all the three curves, and the gap is almost constant, which is about 4 FO4 switching power consumption. As explained in Section II.B, Ling adders take three advantages; but in the uniform arrival time cases, using lower bit P signals is not significantly useful; faster G signals may be dominated by slow P signals; therefore the majority of improvement is probably due to the less power consumption in the first level (see Fig. 2). We will show more advantage of Ling adders in the next subsection.
- The CPU time increases sharply in the cases with tight delay constraints, which prevents the solver finding optimal solutions for larger cases. For example, for 16-bit mixed-radix adders, it takes only 10 seconds to find the optimal solution with 20 FO4 delay constraint but over 70,000 seconds to obtain the optimal solution with 10 FO4 delay constraint. Hence our method is more suitable to solve instances with moderate delay requirements. Also, this fact motivates us to apply hierarchical design to handle high bit-width applications, which will be introduced in the later part of this section.

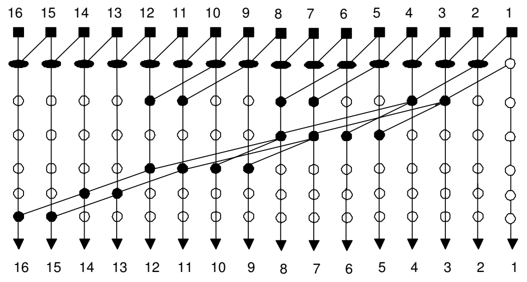


Fig. 7. Minimum Power Radix-2 16-bit Ling Adders

B. Non-uniform Arrival and Required Time

Our formulation is also able to seek for optimal adders with non-uniform signal arrival and required times. This feature is useful in some applications, such as the final adder in a binary multiplier which is used to sum up two partial products, where the middle bits arrive later than most and least significant bits. We design three representative profiles for input arrival: increasing (from least significant to most significant bits), decreasing, and convex (middle bits have large arrival/required values). All experiments are conducted on both normal prefix adders and Ling adders with mixed-radix. The results are shown in Table III. We find that generally Ling adders are able to save power consumption with the same timing constraints from 9.7% to 23.9%; especially in the increasing arrival time case, where the significant bits signals arrive slower, Ling adder can take the advantage of its fast carry computation by using less significant P signals – which means faster P signals, therefore more flexible low power structures could be used, compared with the normal prefix adders.

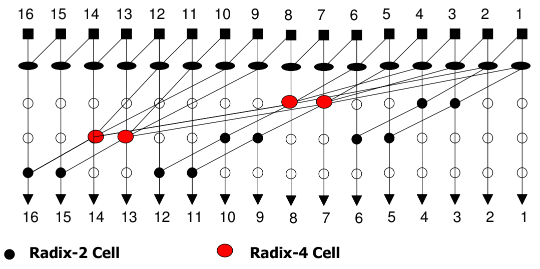


Fig. 8. Minimum Power Radix-2&4 16-bit Ling Adders

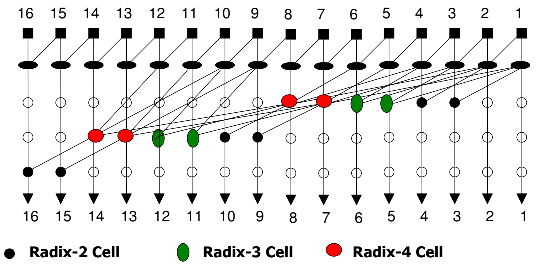


Fig. 9. Minimum Power Mixed-Radix 16-bit Ling Adders

C. Hierarchical Design & ASIC Implementation

To overcome the drawback of unscalable computational effort, we use hierarchical design to handle high bit-width applications. In [9], a two-level hierarchical structure was proposed. However, their structure suffers from a serious weakness that all PG signals produced in the local blocks will be combined in the last stage with the global carry signals, which requires to use a lot of cells therefore consumes much area and power. Hence we proposed a new design, in which only one level is needed. The carry-out signal of each block will be the carry-in signal of the following block and combined in the ILP program to perform the optimization. For example, for a 64-bit adder, we divide the entire structure into 4 blocks, 16-bit for each block. In the ILP block, we will optimize 17-bit adders (except the first block), in which the first bit is the carry-in bit. The carry-in signal may have different arrival time from other bits, since it is generated by the previous block instead of the primary inputs. The schematic diagram is shown in Figure 10.

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%

TABLE III
NON-UNIFORM ARRIVAL/REQUIRED TIME RESULTS

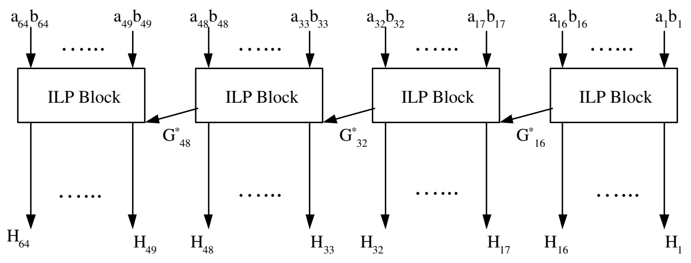


Fig. 10. Hierarchical Design for 64-bit Ling Adders

Method	Area(nm^2)	Delay (ns)	Power (mW)
MC	3512	1.0644	5.471
ILP	3833	0.9425	2.541
ILP	3636	0.9607	2.353
ILP	3114	1.1278	1.973

TABLE IV
64-BIT LING ADDERS ASIC IMPLEMENTATION

To demonstrate the advantage of the proposed ILP methodology, we implement the prefix ing adders by our ILP method in Synopsys Data-path design flow. We wrote a C program to perform the logic synthesis according to the ILP results. The synthesized netlist with relative placement is then placed and routed by Physical Compiler and Astro. According to the necessary physical information including parasitic and coupling capacitance, the area is reported by Physical Compiler, the delay is reported by Astro and the power is reported by Prime Power.

We compare the 64-bit adder obtained by our hierarchical method with fast carry-look-ahead adders generated by Synopsys Module Compiler. The library we use is TSMC 90nm standard cell library. Both the high-radix and Ling adders features are incorporated. The experimental results are shown in Table IV. There are multiple rows for the ILP results because different delay constraints are given. We find that the adder structures generated by ILP can save more than half of the power consumption compared with that generated by MC, with similar area and delay values. We believe that with the implementation of mixed-radix cells, the power consumption could be further reduced.

VI. CONCLUSIONS

This work proposes an integer linear programming formulation to find minimum power prefix Ling adders in with different delay constraints the entire solution space, in which both flexible design choices and practical constraints are incorporated. Our formulation is able to handle different radix and size components. All these considerations are formulated to an integer program with linear constraints and objectives, which can be optimally solved by the ILP solver CPLEX. The experiments show that the optimal structures we find save a lot of power consumption compared with classical designs. We also build hierarchical structures to construct high bit-width adders, which overcome the weakness of unscalable computational time. Our work in fact offers a good framework to find minimum power adders with flexible requirements, since the solver can always produce optimal solutions when different parameters and constraints are plugged in. Hence, it is the prototype of a useful tool for adder designers and able to generate good prefix structure candidates for customized adders.

ACKNOWLEDGEMENT

The authors would like to acknowledge the support of NSF CCF-0618163 and California MICRO Program.

REFERENCES

[1] R. Brent and H. Kung. A regular layout for parallel adders. *IEEE Trans. Computers*, C-31(3):260–264, March 1982.

[2] H. Dao and V. Oklobdzija. application of logical effort on delay analysis of 64-bit static carry-lookahead adder. In *Proc. of* 35^{th} *Asiloma Conference of Singals, Systems and Computers*, volume 2, pages 1322–1324, 2001.

[3] H. Dao and V. Oklobdzija. application of logical effort technqiues for speed optimization and analysis of representative adders. In *Proc. of* 35^{th} *Asiloma Conference of Singals, Systems and Computers*, volume 2, pages 1666–1699, 2001.

[4] G. Dimitrakopoulos and D. Nikolos. High-speed parallel-prefix VLSI Ling adders. *IEEE Trans. Computers*, 54(2):225–231, February 2005.

[5] D. Harris. Logical effort of higher valency adders. In *Proc. of Asilomar Conference of Signals, Systems and Computers*, pages 1358–1362, November 2004.

[6] D. Harris and I. Sutherland. Logical effort of carry progapate adders. In *IEEE Intl. Conf. on Application-Specific Systems, Architectures and Processors*, pages 269–279, 2004.

[7] P. Kogge and H. Stone. A parallel algorithm for the efficient solution of a general class of recurrence relations. *IEEE Trans. Computers*, C-22(8):786–793, August 1973.

[8] H. Ling. High-speed binary adder. *IBM J. R&D*, 25:156–166, May 1981.

[9] J. Liu, Y. Zhu, H. Zhu, J. Lillis, and C. K. Cheng. Optimum prefix adders in a comprehensive area, timing and power design space. In *Proc. of ASP-DAC 2007*, 2007.

[10] S. Mathew, M. Anders, R. Krishnamurthy, and S. Borkar. A 4-GHz 130-nm address generation unit with 32-bit sparse-tree adder core. *IEEE Journal of Solid-State Circuits*, 38(5), May 2003.

[11] S. Naffziger. A subnanosecond 0.5 μm 64b adder design. In *Proc. of Intl. Solid-state Circuits Conf.*, pages 362–363, 1996.

[12] J. Sklansky. Conditional-sum addition logic. *IRE Trans. Electronic Computers*, EC-9:226–231, June 1960.

[13] S. Vanichayobon, S. Dhall, S. Lakshmivarahan, and J. Antonio. Power-speed trade-off in parallel prefix circuits. In *Proc. of SPIE*, volume 4863, pages 109–120, 2002.

[14] H. Zhu, C. K. Cheng, and R. Graham. Constructing zero-deficiency parallel prefix adder of minimum depth. In *Proc. of ASP-DAC 2005*, pages 883–888, 2005.

[15] R. Zimmermann. Non-heuristic optimization and synthesis of parallel prefix adders. In *Proc. of In. Workshop on Logic and Architecture Synthesis*, pages 123–132, 1996.