An Interconnect-Centric Approach to Cyclic Shifter Design Using Fanout Splitting and Cell Order Optimization

Haikun Zhu, Yi Zhu, Chung-Kuan Cheng
CSE Dept., UCSD
La Jolla, CA 92093-0404, USA
Email: {hazhu,y2zhu,kuan}@cs.ucsd.edu

David M. Harris
Harvey Mudd Colledge
Claremont, CA 91711
Email: David_Harris@hmc.edu

***Abstract*— We propose two orthogonal approaches to logarithmic cyclic shifter design. The first method, called fanout splitting, replaces multiplexers in a conventional design with demultiplexers which have two fanouts driving the shifting and non-shifting paths separately. The use of demultiplexers has a two-fold effect; it cuts the accumulated wire load on the critical path from $O(N \log_2(N))$ to $O(N)$, and reduces the switching probabilities on the inter-stage long wires from 1/4 to 3/16. We then perform cell order optimization to further improve the delay, and formulate it as an integer linear programming problem. For the 64-bit case, the two approaches together reduce the total delay by 67.1% and dynamic power consumption by 17.6%, respectively.**

General Terms: Performance, Power, Design
Keywords: cyclic shifter, interconnect, fanout splitting, permutation, integer linear programming

1. INTRODUCTION

Binary shifters, similar to adders and multipliers, are indispensable in high performance microprocessors, especially in those that support floating-point operations. For communication applications such as encryption and error control coding, the cyclic shifter is a critical component because rotation operations are enormously needed. Yet the simplicity of the shifter logic misleadingly disguises its importance in circuit design; Literature on shifter design is relatively scarce compared to that of adders and multipliers, and textbooks typically cover shifter in just one or two pages [1], [2]. The main reason is that the complexity of the shifters comes from the internal wire connections which does not fit into the traditional logic-centric design methodology.

It is well-known that in terms of design style there are two types of shifters for circuit designers to choose from: *array shifter* or *logarithmic shifter* [1]. The former is noted for its high speed because in theory every data signal only passes through one transmission gate. However, in practice the capacitance presented to the input signals increases linearly with the word length, and the number of transistors grows quadratically with the word length. Moreover, for array shifter an additional decoder for control signals is required. These factors make the array shifter less appealing for large word length. As an attractive alternative, the logarithmic shifter completes the shifting in $log_2(N)$ stages, where $N = 2^n$ is the word length. At each stage the data signals either uniformly pass through or shift by 2^i bits, where $0 \leq i \leq n-1$. However, the logarithmic shifter has its own weakness. The amount of inter-stage interconnect wires roughly double from one stage to the next. The situation is even aggravated with the aggressive technology scaling due to which the interconnect is dominant for both delay and power consumption.

There were few attempts to improve the shifter by optimizing the interconnect wires. M. A. Hillebrand et al. [3] proposed to half the wire length in a cyclic shifter (rotator) by permuting the cell positions in the intermediate stages. In [4], a two-dimensional folding strategy is suggested for the layout of a cyclic shifter. However, since the shifter is usually aligned with the adder and multiplier in a datapath module and can not decide the module width on its own, the approach is hardly useful in practice. At the logic level, ternary shifting [5] is proposed to replace binary shifting so that the number of stages is reduced. In essence, this method uses 3-to-1 MUXes (multiplexer) to build the shifter network. In [6] Yih et al. proposed a multilevel approach to reduce the number of transmission gates in the barrel shifter.

We propose two methods to alleviate the interconnect burden in the logarithmic cyclic shifter (rotator). Our main contributions can be summarized as follows.

- We propose fanout splitting to decouple the shifting and non-shifting behaviors at each stage so that the non-shifting paths can be saved from switching when the data signals are configured to pass through, and vice versa. In practice, this idea is realized by replacing the MUXes in a traditional design with DEMUXes (demultiplexer) which have two outputs governing the shifting and non-shifting paths separately, giving rise to the term "fanout splitting". We show that by doing so the accumulated wire load on the critical path is reduced from $O(N \log_2(N))$ to $O(N)$. Moreover, the switching probabilities on the inter-stage wires of the new design is lower than that of the conventional MUX-based design, resulting in smaller overall switched capacitance.
- We apply cell order optimization to our DEMUX-based design to further reduce the delay. We formulate the optimization as an Integer Linear Programming (ILP) problem and solve it using commercial ILP solver CPLEX 9.1. For 32- and 64-bit cases a sliding window heuristic is devised to tackle the complexity issue.
- Using the ILP formulation framework we also study the power-delay tradeoff for cell ordering. We show that linear order placement is inherently inefficient in terms of longest path delay.

The experimental results show that for 64-bit case the total delay is reduced by 67.1%, and the dynamic power consumption is reduced by 17.6%, when both fanout splitting and cell order optimization are applied.

The rest of the paper is organized as follows: In Section 2 we formally state the shifter design problem. Section 3 gives the motivation of our work and describes the fanout splitting technique in detail. The cell order optimization and its ILP formulation are presented in Section 4. In Section 5 we give the experiment results. Section 6 concludes the paper.

2. NOTATIONS AND PROBLEM STATEMENT

We confine our discussion to the logarithmic cyclic shifter. In the rest of the paper, "shifter" refers to the logarithmic cyclic shifter unless otherwise noted. We may sometimes use rotator interchangeably.

A shifter takes as inputs an N-bit binary data word $\mathrm{D}[\mathrm{N}-1:0]$ and an n-bit control word $\mathrm{S}[n-1:0]$, and produces an output word $\mathrm{Z}[\mathrm{N}-1:0]$ with $\mathrm{N} = 2^n$ for some positive integer n. The binary value of the control word is denoted by $|\mathrm{S}| := \sum_{i=0}^{n-1} \mathrm{S}[i] \cdot 2^i$. The

1-4244-0630-7/07/$20.00 ©2007 IEEE.

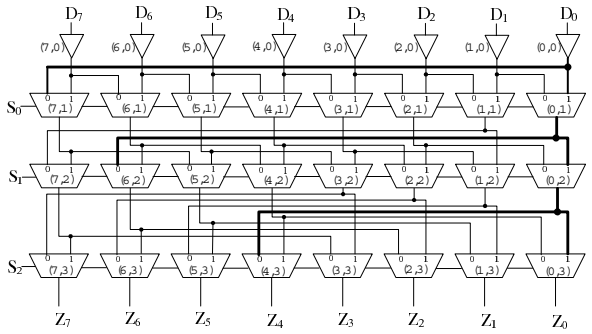

Fig. 1. 8-bit conventional MUX-based shifter.

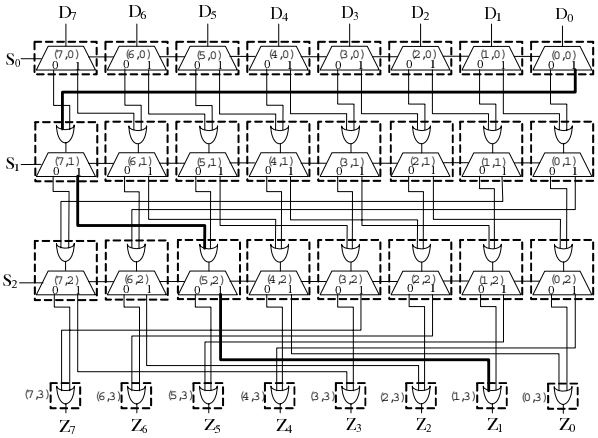

Fig. 2. Proposed DEMUX-based (fanout splitting) shifter.

shifter implements the function ROTATE:

$$\begin{aligned} \mathrm{Z[N-1:0]} &= \textsc{rotate}(\mathrm{D[N-1:0]}, \mathrm{S}[n-1]) \\ &:= (\mathrm{D}[|\mathrm{S}|-1:0], \mathrm{D[N-1:|S|]}) \end{aligned}$$

where (A, B) represents the concatenation of two binary strings A and B.

Since the logarithmic shifter is an interconnect intensive component, the main design objective is to come up with a scheme that reduces both power dissipation and delay induced by the wires.

3. FANOUT SPLITTING SHIFTER DESIGN

3.1. Motivation

Fig. 1 shows an example of a conventional 8-bit rotator using MUXes (hereafter referred to as MUXSHIFTER). The MUXes are arranged in an array where the MSB cells are on the left. Assuming there is an input buffer stage, the whole MUX network including the input buffers can be mapped onto a grid graph $G = (V, E)$ where $V := \{(column, row) = (i, j) | (i, j) \in \langle 0, 1, ..., N-1\rangle \times \langle 0, ..., n\rangle\}$. From each node (i, j) with $j < n$ there are two outgoing edges: $(i, j) \rightarrow (i, j+1)$ and $(i, j) \rightarrow (i - 2^j \mod N, j+1)$. Following the terminology used in [3], we call $(i, j+1)$ the *unshifted child* and $(i-2^j \mod N, j+1)$ the *shifted child* of (i, j). Likewise, each node (i, j) with $j > 0$ has two incoming edges from $(i, j-1)$ and $(i + 2^j \mod N, j-1)$, the former called the *unshifted parent* and the latter called the *shifted parent*.

The issue with MUXSHIFTER is that the way the MUXes are interconnected is unaware of the functionality of the shifter. The two edges coming out of node (i, j) are electrically hard-wired together. In the grid graph, this can be modeled by assigning weight $2^j + 1$ to both $(i, j) \rightarrow (i, j+1)$ and $(i, j) \rightarrow (i - 2^j \mod N, j+1)$. As a result, when the shifter is configured to shift 0 bit, all the lateral wires have to be switched as well, unnecessarily consuming more power. This is clearly against the intuitive switch-only-when-needed low power design principle.

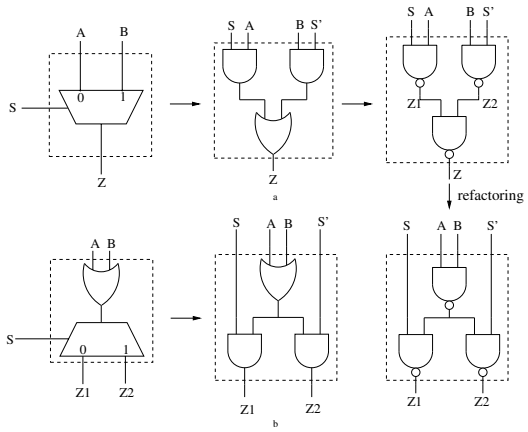

Fig. 3. The gate level implementation of (a) MUX; and (b) OR+DEMUX (cross switch).

Even worse, the longest lateral wires at each stage in MUXSHIFTER are accumulated on the critical path. Consider the path from D_0 to Z_0 in Fig. 1. The accumulated wire load (highlighted by heavy lines) on this path is 20, assuming 1 unit wire length per column and per row. In general, the accumulated wire load on the critical path for MUXSHIFTER is $\Omega(N \log_2(N))$ [3] where N is the word length.

3.2. Fanout Splitting Approach

Motivated by the above observations, we propose to use DEMUXes to build the shifter in lieu of the MUXes in conventional designs. Each DEMUX has two outputs, the left one is the unshifted signal while the right one is the shifted signal. At the next stage, we precede each DEMUX with an OR gate which collects the shifted and unshifted signals from the previous stage. Such a design is shown in Fig. 2. Since a DEMUX has separate outputs, this technique is termed "fanout splitting". We call the new design DEMUXSHIFTER.

Similar to MUXSHIFTER, DEMUXSHIFTER can be represented by exactly the same grid graph. The difference is that the two edges coming out of node (i, j) are now independent. Edge $(i, j) \rightarrow (i, j+1)$ has weight 1 while edge $(i, j) \rightarrow (i-2^j \mod N, j+1)$ has weight $2^j + 1$.

3.2.1. Impact on Longest Delay Path

Because of fanout splitting, the shifting and non-shifting behaviors are now decoupled. This directly reduces the accumulated wire load on the critical path. For the 8-bit case shown in Fig. 2, the wire load of the critical path is now 16. In fact, we have the following conclusion.

Claim 1: The wire length of the critical path in DEMUXSHIFTER is $O(N)$.

Proof: We show a lower bound for the critical path wire length. From level j to $j+1$ there are two types of horizontal wires. Those go to the lower bits have length 2^j. Those wrap around to the higher bits have length $N - 2^j$. The key observation is that any path from a topmost node $(i_1, 0)$ to a bottommost node (i_2, n) shall contain at most one horizontal wire that wraps around. Therefore,

$$\begin{aligned} &\text{critical path wire length} \\ &\leq \max_{0 \leq j \leq n-1} \left\{ \Big(\sum_{\substack{k=0 \\ k \neq j}}^{n-1} 2^k \Big) + N - 2^j + n \right\} \\ &= \max_{0 \leq j \leq n-1} \left\{ \Big(\sum_{k=0}^{n-1} 2^k \Big) + N - 2^{j+1} + n \right\} \\ &\leq 2N - 3 + n \end{aligned}$$

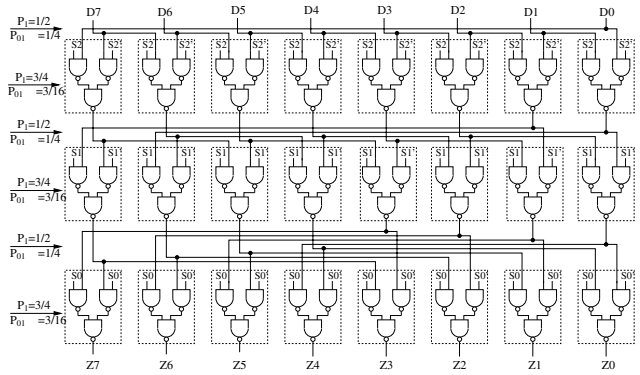

Fig. 4. 8-bit MUXSHIFTER using NAND gates.

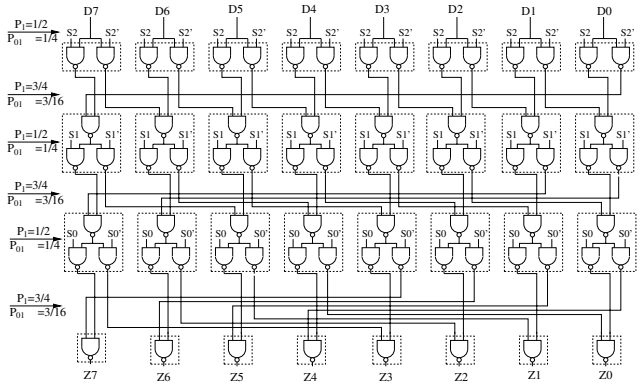

Fig. 5. 8-bit DEMUXSHIFTER using NAND gates.

This proves the claim. ■

To understand the gate complexity, we give the gate implementations of the MUX and the OR+DEMUX conglomerate (essentially a 2-input/2-output switch box) in Fig. 3. Interestingly, at gate level the fanout splitting design is really like re-factoring the OR function in the MUXes later to be combined with the logic of the next stage. Note the NAND gate version of the cross switch does not really implement a cross switch; it is derived from the NAND gate version of the MUX by utilizing the re-factoring observation. Fig. 4 and Fig. 5 illustrate the NAND gate implementations of MUXSHIFTER and DEMUXSHIFTER, respectively. From the figures we see the fundamental difference between MUXSHIFTER and DEMUXSHIFTER is the organization of the inter-stage wires. They both have the same gate complexity $O(N \log_2(N))$.

3.2.2. Impact on Dynamic Power Consumption

At a first glance, the total inter-stage wire length in DEMUXSHIFTER is the same or even a little bit more than that of MUXSHIFTER. Nevertheless, as we will show in the next, the switching probability on the wires decreases from $1/4$ to $3/16$, which leads to less overall power consumption.

It is well-known that the dynamic power consumption of a circuit is proportional to the effective switched capacitance C_{eff},

$$P_{\text{dynamic}} = C_{\text{eff}} V_{\text{supply}}^2/2 = (C_{\text{load}} P_{0\to1}) V_{\text{supply}}^2/2 \quad (1)$$

where $P_{0\to1}$ is the switching probability on the load capacitance. In general, predicting the signal switching probabilities in a circuit is difficult because of signal correlation due to re-convergent paths. However, the shifter is a strictly levelized design and does not contain re-convergent fanouts, hence its switching probabilities can be easily calculated.

Define the 1-probability P_X to be the probability that a signal X is *one*. The $0 \to 1$ switching probability of signal X can be written as

$$P_{X|0\to1} = P_X(1 - P_X) \quad (2)$$

TABLE I
TRUTH TABLE OF THE MUX USING NAND GATES

S	S'	A	B	Z1	Z2	Z
0	1	0	0	1	1	0
0	1	0	1	1	0	1
0	1	1	0	1	1	0
0	1	1	1	1	0	1
1	0	0	0	1	1	0
1	0	0	1	1	1	0
1	0	1	0	0	1	1
1	0	1	1	0	1	1

We start by assuming that all the primary input signals $[D_7{:}D_0]$, $[S_2{:}S_0]$ are equal probable at 0 or 1. Consider the NAND gate implementation of a MUX in Fig. 3(a). From the truth table (Table I) we see that

$$P_{Z1} = P_{Z2} = 3/4; P_Z = 1/2$$

We then propagate the 1-probabilities downward across the shifter network, and calculate the switching probabilities using Eqn. (2). The MUXSHIFTER (Fig. 4) and DEMUXSHIFTER (Fig. 5) are annotated with both 1-probabilities and switching probabilities. From the figures we clearly see that DEMUXSHIFTER has lower switching probabilities on the inter-stage long wires.

4. CELL ORDER OPTIMIZATION BY INTEGER LINEAR PROGRAMMING

M. A. Hillebrand et al. [3] observed that the longest delay path in a cyclic logarithmic shifter can be reduced by permuting the cells within each row in the intermediate stages, and they applied this technique to a MUX-based design. Apparently, our DEMUX-based shifter can benefit from this technique as well. Different from their constructive approach, we formulate the cell permutation optimization as an Integer Linear Programming (ILP) problem, and use the state-of-the-art CPLEX solver to obtain the results.

4.1. ILP Formulation

Assuming the word length $N = 2^n$, an N-bit DEMUXSHIFTER consists of $n + 1$ levels. Level 0 is the input stage while level n is the output stage (see Fig. 5). The cell order of the input and output stages are fixed to be linear with MSB on the left and LSB on the right. For each intermediate stage, a set of binary decision variables specifying the permutation order is introduced.

- $x_{ij}^l \in \{0, 1\}$: 1 if and only if logic cell i is placed at physical position j on level l $(0 \le i, j \le N-1, 1 \le l \le n-1)$.

Since each logic cell occupies one and only one physical position, the following constraints naturally hold:

$$\sum_{i=0}^{N-1} x_{ij}^l = 1 \quad (0 \le j \le N-1, 1 \le l \le n-1) \quad (3)$$

$$\sum_{j=0}^{N-1} x_{ij}^l = 1 \quad (0 \le i \le N-1, 1 \le l \le n-1) \quad (4)$$

Note that the set of decision variables x_{ij}^l and constraints (3), (4) completely define the permutation solution space. The rest of the problem is to represent the objective, i.e., minimizing the longest path.

Denote a cell with logic index i at level l as C_i^l. Pick a cell C_i^0 at the input stage and a cell C_j^n at the output stage, there is an unique delay path from C_i^0 to C_j^n. Put another way, from cell C_j^n there are exactly N path tracing back to the input level. The scenario for the 8-bit case is shown in Fig. 6. The logic indices of the cells on each delay path are annotated on the graph, while the physical positions of the cells are for illustration purpose only — they can be anywhere in their respective rows. Thus for an N-bit shifter, there are in total

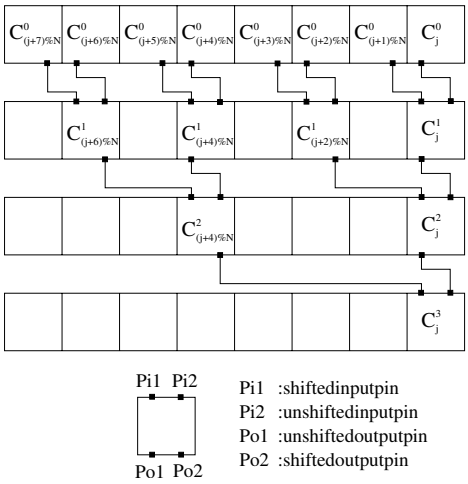

Fig. 6. Delay paths ending at C_j^3.

N^2 delay paths. The length of the longest path, call it $T_{\max}$, is given by

$$T_{\max} = \max\{\text{length of the delay path from } C_i^0 \text{ to } C_j^n | 0 \le i, j \le N\} \quad (5)$$

which can be expanded into N^2 linear constraints as

$$T_{\max} \ge \text{length of the delay path from } C_i^0 \text{ to } C_j^n \quad \text{for } 0 \le i, j \le N-1 \quad (6)$$

with the objective

$$\text{obj}: \ \text{minimize } \mathrm{T_{max}} \quad (7)$$

Each input-to-output delay path consists of n wire segments. We show the technique to represent the length of a single wire segment as a set of linear constraints. Without loss of generality, suppose $C_{i_1}^0$ is connected to $C_{i_2}^1$, yet their respective physical locations are unknown. The length of the wire segment connecting them can be written as

$$d = \Big| \sum_{j=0}^{N-1} j \cdot x_{i_1 j}^0 - \sum_{j=0}^{N-1} j \cdot x_{i_2 j}^1 \Big| \quad (8)$$

which can be converted into linear constraints

$$d - \sum_{j=0}^{N-1} j \cdot x_{i_1 j}^0 + \sum_{j=0}^{N-1} j \cdot x_{i_2 j}^1 \ge 0 \quad (9)$$

$$d + \sum_{j=0}^{N-1} j \cdot x_{i_1 j}^0 - \sum_{j=0}^{N-1} j \cdot x_{i_2 j}^1 \ge 0 \quad (10)$$

Note that constraints (9) and (10) only set the lower bound for d, hence are not entirely equivalent to Eqn. (8). However, since our objective is to minimize $T_{\max}$, and $T_{\max}$ is non-decreasing w.r.t. d, we conclude that the lower bound for d must be achieved in the optimal solutions.

Once we know how to represent the length of a single wire segment, it is straightforward to express the length of an input-to-output delay path. In modelling the delay paths we have neglected the gate delay and vertical wire length, since they equally contribute to all paths.

In a nutshell, our *minimum delay* ILP formulation comprises constraints (3), (4), (6) and objective (7). We also consider the *minimum power* formulation which seeks to minimize the total wire length T_{total} subject to a given constraint on maximum delay.

$$\text{obj}: \ \text{minimize } \mathrm{T_{total}} \quad (11)$$

$$\text{s.t.} \quad \mathrm{T_{max}} \le \text{Constant} \quad (12)$$

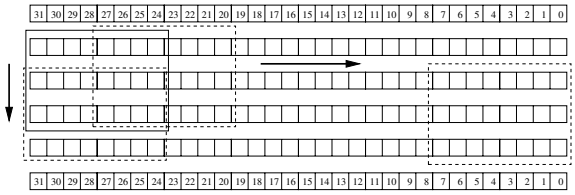

Fig. 7. Sliding window scheme for the 32-bit case.

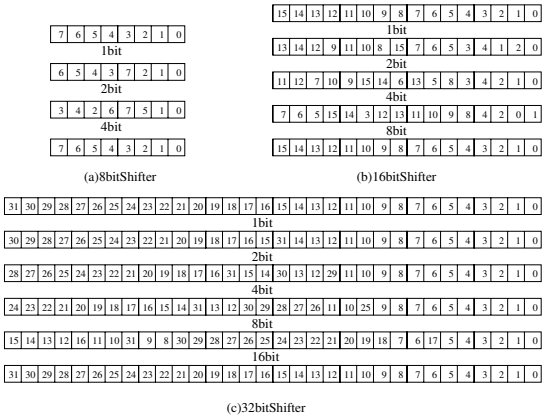

Fig. 8. Optimal cell order for minimum delay.

The minimum power formulation helps to study the tradeoff between delay and power, since total wire length is an indicator of the dynamic power consumption.

4.2. Sliding Window Scheme

The CPLEX solver uses a branch and bound method with linear relaxation [7], thus the running time greatly depends on the problem size. In our ILP formulation, the number of decision variables is $(n-1)N^2$, which grows more than quadratically. As a result, the 8-bit case was solved in seconds while the 16-bit case took almost one day to find an optimal solution. For 32- and 64-bit cases, treating the problem as a whole becomes infeasible. We now describe a sliding window heuristic to tackle the complexity issue of our ILP formulation.

The sliding window scheme optimizes the shifter in multiple passes. In each pass, a sliding window is superimposed on the shifter layout, as shown in Fig. 7. Only cells within the sliding window are allowed to permute freely, with the goal to minimize $T_{\max}$ (min-delay formulation) or T_{total} (min-power formulation). The window then moves from left to right and top to bottom until it reaches the bottom-right corner. The procedure terminates when there is no improvement between two passes.

Within each pass four parameters control the optimization process: window width WW, window height WH, horizontal sliding step HS and vertical sliding step VS. Tradeoff must be made to balance the runtime and result quality when selecting these parameters. In our experiments we have empirically found that the following set of parameters to be satisfying:

- WW = 8 columns
- WH = 3 rows
- HS = 4 columns
- VS = 1 row

In particular, HS is set to 4 columns so that the cells have large chance to move across the entire row; and WH is set to 3 rows to allow adequate interaction between rows, thus avoiding being trapped in the local minima. The same set of parameters is used for both 32- and 64-bit cases.

Fig. 8 shows delay-optimal cell orders for 8-bit, 16-bit and 32-bit cases. Note the solutions for 8-bit and 16-bit are global optimum

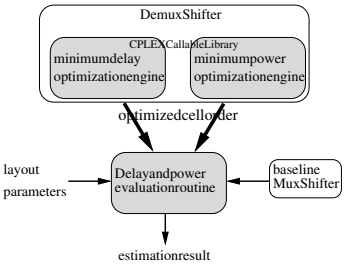

Fig. 9. Estimation flow.

TABLE II
DEVICE/WIRE PARAMETERS

2λ	0.18	0.15	0.13	0.10	0.07
C_g	0.234	0.220	0.135	0.072	0.066
C_a	0.0589	0.0540	0.0462	0.0720	0.0611
C_f	0.0302	0.0248	0.0183	0.0141	0.0148
C_x	0.0583	0.0494	0.0428	0.0453	0.0416
h_w	0.63	0.92	1.06	1.54	1.03

2λ(μ m): technology feature size;
C_g(fF): input capacitance of minimum size inverter;
C_a(fF/μm^2): unit area capacitance of wire;
C_f(fF/μm): unit fringing capacitance of wire;
C_x(fF/μm): unit coupling capacitance of wire.
Note: All wire capacitance are obtained assuming 2x minimum width and 2x minimum spacing.

while the solution for 32-bit is suboptimal with the sliding window scheme. The result for 64-bit case is not shown here for it is too small to display.

5. DELAY AND POWER ANALYSIS

5.1. Estimation Flow

We implemented the sliding window scheme in C using the CPLEX Callable Library [8]. For both min-delay formuation and min-power formulation, the optimized cell order is fed into a delay and power evaluation routine. The evaluation routine takes in a set of parameters based on a technology independent layout model, which we will describe in the next, and produces the estimation results. The whole process is illustrated in Fig. 9.

5.2. Technology Independent Layout Model

We use the logical effort method [9] for fast, technology independent delay/power estimation for both MUXSHIFTER and DEMUXSHIFTER. For a single stage gate, logical effort measures its delay in unit of τ, the delay of an ideal inverter with no parasitic driving an identical inverter.

$$D = D_{\text{abs}}/\tau = gh + p \tag{13}$$

where g is the *logical effort* of the gate, which is defined as the ratio of the input gate capacitance to the input capacitance of an inverter with the same unit effective resistance; h is the *electrical effort* of the gate, that is, the ratio of load capacitance to input capacitance. p characterizes the parasitic (intrinsic) delay of the gate. For a static CMOS NAND gate, $g = 4/3$ and $p = 2$.

We consider the wire load effect by incorporating wire capacitance into h. Thus the electrical effort of a NAND gate is written as

$$h = h_{\text{fanout}} + h_w \cdot l_w \tag{14}$$

where h_{fanout} is the number of load gates and l_w is the length of the driven net normalized to the width of a MUX/DEMUX cell, or simply the number of columns the wire spanned. Natually, h_w is the electrical effort contributed by the wire per column spanned.

Assuming all the NAND gates in the shifter are 2x of the minimum size uniformly and the MUX/DEMUX cell width is 80λ, we directly

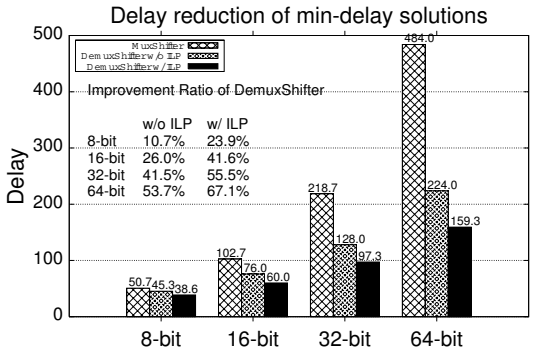

Fig. 10. Delay comparison of MUXSHIFTER and DEMUXSHIFTER; in unit of τ (intrinsic delay of a minimum sized inverter).

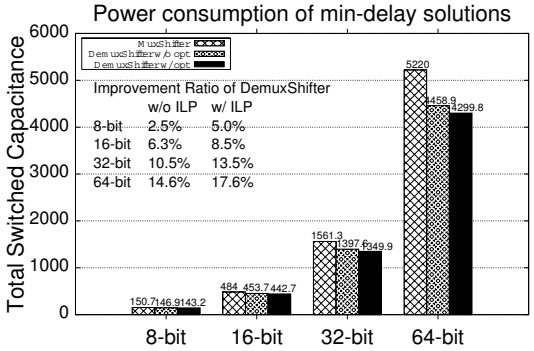

Fig. 11. Power comparison of MUXSHIFTER and DEMUXSHIFTER; in unit of C_g (input capacitance of a minimum sized inverter).

borrow the technology parameters from [10], and use the following formula to estimate h_w for each technology node:

$$\begin{aligned} h_w &= \frac{\text{Capacitance of wire per column spanned}}{\text{Input capacitance of a 2x NAND gate}} \\ &= \frac{(C_a \cdot 4\lambda + C_f + C_x) \cdot 80\lambda}{C_g \cdot 2 \cdot 4/3} \end{aligned} \tag{15}$$

From Table II we see that it is safe to assume $h_w = 1$ across different technology nodes, as it is used in [11].

Note that in the analysis we have assumed that the resistive RC delay of the wires is negligible. This assumption is supported by the work of Huang and Ercegovac [10].

The overall path delay is then simply the sum of the stage delays:

$$D_{\text{path}} = \sum_i D_i = \sum_i g_i h_i + \sum_i p_i \tag{16}$$

Since the shifter does not contain re-convergent paths we can simply search the N^2 input-to-output paths and pick the longest one.

The dynamic power is estimated by summing up all the switched capacitance C_{eff} in the shifter, including gate input capacitance and wire capacitance. The input gate capacitance of a 2x NAND gate is $8C_g/3$, which is also the capacitance of the wire of one column span.

5.3. Results

Fig. 10 shows the delay estimation results. Comparing to conventional MUXSHIFTER, the DEMUXSHIFTER without cell order optimization reduces the delay by 41.5% for 32-bit case and 53.7% for 64-bit case. When cell order optimization is applied, the reduction further increases to 55.5% and 67.1%, respectively.

Fig. 11 shows the power estimation results. For 32-bit and 64-bit cases, the dynamic power improvement is 13.5% and 17.6%, respectively. This is less than the reduction of switching probabilities on the wires (from 1/4 to 3/16) because the gate power is also included.

To study the power-delay tradeoff in the shifters, we also solved the min-power ILP problem (Eqn. (11) and Eqn. (12)) for our proposed

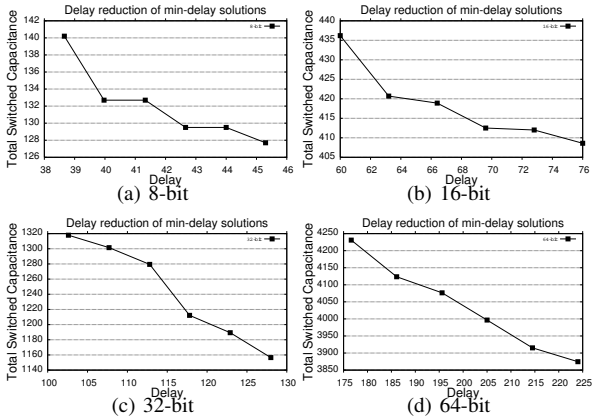

Fig. 12. Power-delay tradeoff; power is in unit of C_g and delay is in unit of τ.

DEMUXSHIFTER, and the results are shown in Fig. 12. It seems that the power (total switched capacitance) is much harder to optimize. For example, in the 64-bit case relaxing the delay constraint from 176.6τ to 224τ only results in 8.4% reduction in power. Nevertheless, the results make sense, because intuitively the shifting path and non-shifting path could be conflict objectives when moving a cell. This indicates that the linear cell order is inherently inferior in terms of worst case delay.

Note that in all the experiments, we have assumed uniform sizing for the DEMUXSHIFTER. In fact, one can size up the NAND gate that drives the longer wire in each DEMUX cell to further improve the delay. This enhancement is applicable to both the permuted and non-permuted DEMUXSHIFTER, and it is interesting to study which scheme benefits more from sizing.

6. CONCLUSIONS AND FUTURE WORK

We presented two orthogonal methods to improve the critical path delay and power consumption in logarithmic cyclic shifters. The first method, called fanout splitting, relies on using DEMUXes to reduce accumulated wire load on the critical path, as well as the switching probabilities of the inter-stage wires. We then optimize the cell order of the intermediate stages by integer linear programming (ILP) to further reduce the delay. Using the ILP formulation we also study the delay-power tradeoff in the shifter, and our results show that linear cell placement is inherently inferior in terms of critical path delay.

For simplicity, the delay and power analysis in this paper have assumed a logical effort based approach, which ignores the dynamic characteristics of the circuits such as glitching power. To remedy this deficiency an extensive layout-extraction-simulation design flow is desired, as conducted in [12]. Nevertheless, our estimation is meaningful in that it provides a fast high level comparison.

Another possible future work is to extend fanout splitting and cell order optimization to the ternary shifter design [5].

ACKNOWLEDGMENT

The work is supported in part by National Science Foundation under the agreement number CCF-0618163.

REFERENCES

[1] N. H. Weste and D. Harris, *CMOS VLSI Design : A Circuits and Systems Perspective*, 3rd ed. Addison-Wesley Publisher, May 2004.

[2] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits - A Design Perspective*, 2nd ed. Prentice-Hall Publisher, Dec. 2002.

[3] M. A. Hillebrand, T. Schurger, and P.-M. Seidel, "How to half wire lengths in the layout of cyclic shifter," in *Proc. of the Intl. Conf. on VLSI Design*, 2001, pp. 339–344.

[4] P.-M. Seidel and K. Fazel, "Two-dimensional folding strategies for improved layouts of cyclic shifters," in *Proc. of the IEEE annual Symposium on VLSI*, 2004, pp. 277–278.

[5] G. M. Tharakan and S. M. Kang, "A new design of a fast barrel switch network," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 2, pp. 217–221, 1992.

[6] S.-J. Yih, M. Cheng, and W.-S. Feng, "Multilevel barrel shifter for cordic design," *Electronics Letters*, vol. 32, no. 13, pp. 1178–1179, June 1996.

[7] *ILOG CPLEX 9.1 Reference Manual*, ILOG Inc., Apr. 2005.

[8] *ILOG CPLEX Callable Library 9.1 Reference Manual*, ILOG Inc., Apr. 2005.

[9] I. Sutherland, B. Sproull, and D. Harris, *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann, Feb. 1999.

[10] Z. Huang and M. Ercegovac, "Effect of wire delay on the design of prefix adders in deep submicron technology," in *Proc. of the Asilomar Conf. on Signals Systems and Computers*, vol. 2, 2000, pp. 1713–1717.

[11] D. Harris and I. Sutherland, "Logical effort of carry propagate adders," in *Proc. of the Asilomar Conf. on Signals Systems and Computers*, vol. 1, 2003, pp. 873–878.

[12] K. P. Acken, M. J. Irwin, and R. M. Owens, "Power comparisons for barrel shifter," in *Proc. of the Intl. Symp. on Low Power Electronics and Design (ISLPED)*, 1996, pp. 209–212.