Implementation of Pulsed-Latch and Pulsed-Register Circuits to Minimize Clocking Power

Seungwhun Paik
Dept. of Electrical Engineering, KAIST
Daejeon 305-701, Korea

Gi-Joon Nam
IBM Austin Research Laboratory
Austin, Texas

Youngsoo Shin
Dept. of Electrical Engineering, KAIST
Daejeon 305- 701, Korea

ABSTRACT

A pulsed-latch can be modeled as a fast flip-flop. This allows conventional flip-flop designs to be migrated to pulsed-latch versions by simple replacement to reduce the clocking power. A key step in the migration process is to insert pulsers, which generate clock pulse to drive local latches; the number of pulsers as well as the wirelength of clock routing must be minimized to reduce the clocking power. We formulate a pulser insertion problem to find a set of latch groups where each group shares a pulser and its load constraint is satisfied; both an ILP formulation and a heuristic algorithm are presented to solve the problem. Experimental results of circuits implemented with 32-nm CMOS technology show that the clocking power of pulsed-latch designs obtained by our approach is 5.9% less than that of greedy approach; this is 44.7% less than that of flip-flop designs. We also consider the problem of pulsed-register where a pulser is integrated with multiple latches. A concept of logical distance is explored during our clustering algorithm to minimize the overhead of signal wirelength when converting flip-flops to pulsed-registers. Compared with flip-flop circuits, signal wirelength is increased by 6.3%, which is 1.4% smaller than without considering logical distance, while reducing the clocking power by 24%.

Categories and Subject Descriptors: B.7.1 [**Integrated Circuits**]: Types and Design Styles—*VLSI*; J.6 [**Computer-Aided Engineering**]: Computer-Aided Design

General Terms: Algorithms

Keywords: Pulsed-latch, pulsed-register, low clocking power

1. INTRODUCTION

Flip-flops are the most popular sequencing elements (SEs) that are used in almost every digital circuit. However, flip-flops take an appreciable portion of clock period and total power consumption. In modern ASIC designs, clock network is responsible for 40–50% of total dynamic power consumption; a significant portion, e.g. 47% [1], of the clocking power is consumed by flip-flops. Since it is difficult to control the extent of using flip-flops in a design, exploring an alternative SE is a viable solution to reduce the clocking power without significantly altering the clock distribution.

A pulsed-latch, which is a latch driven by a narrow clock pulse, is a promising alternative to the conventional flip-flop to reduce the power consumption. A pulsed-latch can be built from any transparent latch by supplying a clock pulse. The clock pulse can be generated by a pulse generator (or a pulser), which takes a normal clock with 50% duty cycle as an input. Pulsed-latches inherit the small sequencing overhead of a latch and has some tolerance to clock skew and jitter due to time-borrowing; the increased timing slack can be used to reduce the power consumption by employing low-power techniques such as gate sizing, multiple V_{dd}, and etc. In ASICs, the small amount of time-borrowing can be deliberately ignored to model the pulsed-latch as a fast flip-flop; this allows a simple replacement of flip-flops with pulsed-latches. Pulsed-latches have fewer transistors that are triggered by clock signal than flip-flops do; so using them benefits an appreciable amount of clocking power.

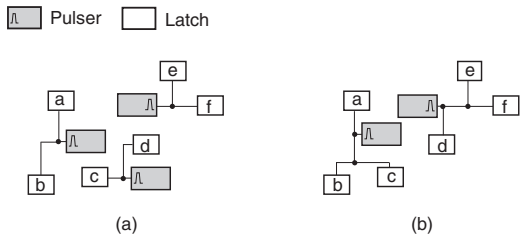

Figure 1: (a) Grouping c and d results in three latch groups while (b) there is a better solution with two groups.

To obtain pulsed-latch circuits, we first substitute all (or some) flip-flops with latches [2, 3]; delay buffers have to be inserted to fix a likely increase of hold time violations [4]. This migration alone achieves 20% of power saving [2] and 10% boost in clock frequency [4]. A key step in the migration process is to insert pulsers; we need to identify groups of latches so that each group can be connected to a single pulser. Grouping of latches determines the capacitance of wire to route clock pulse, which again affects the number of latch groups. This is illustrated using an example in Figure 1(a); once c and d form a latch group, no more latches can be added to the group due to a maximum load that a pulser can drive; this results in using three pulsers to drive all latches whereas there is a better solution with less clocking power that uses one less pulser and shorter total wirelength of clock routing as shown in Figure 1(b). Therefore, we try to find a grouping of latches such that the number of latch groups (or pulsers) is minimized while the load capacitance of each group is less than a given maximum; the former is to minimize the power consumption as pulsers contribute large portion of total power [5] and the latter is to ensure the correct pulse shape [6]. This problem is addressed in Section 3.

A pulser may be embedded in a latch [7], called a pulsed flip-flop, to avoid the burden of routing clock signal from a pulser to its latch. The integration of pulser and latch comes at the obvious cost of more area and power consumption than using an external pulser that are shared by multiple latches, but the benefit of pulsed-latch in less sequencing overhead (i.e., sum of clock-to-Q delay and setup time) and simple timing model is maintained. To take advantage of both pulsed-latch (shared pulser) and pulsed flip-flop (no distortion of pulse shape), we may integrate more than one latch with a single pulser, which yields a pulsed-register. Then the problem is to find a group of latches to form a pulsed-register without degrading the

978-1-4577-1400-9/11/$26.00 ©2011 IEEE

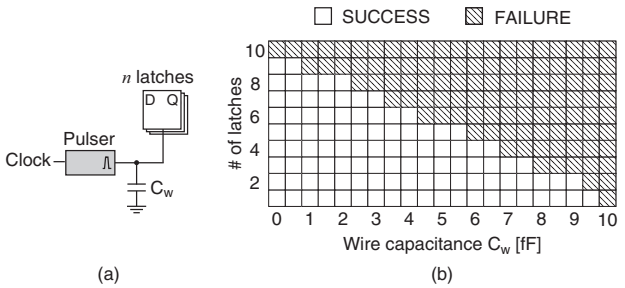

Figure 2: (a) Experimental setup to test the over-loading of pulsers and (b) a shmoo plot.

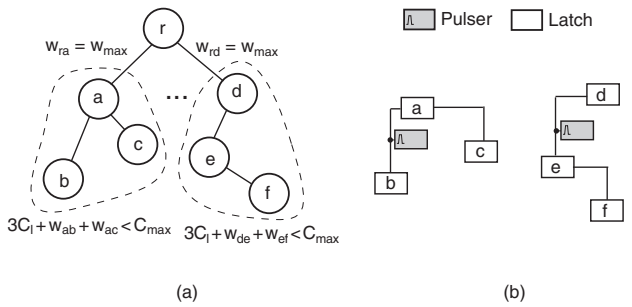

Figure 3: (a) An example MST and (b) corresponding pulser groups.

quality of a conventional placement too much; this is addressed in Section 4.

The main contributions of this paper are as follows.

- Formulation of the pulser insertion problem to minimize both the number of pulsers and the wirelength of clock routing to minimize clocking power; an optimal ILP formulation and a graph-based heuristic algorithm to solve the problem (Section 3).
- Formulation of the pulsed-register problem; a concept of logical distance is introduced to identify flip-flop groups while minimizing impact on signal wirelength (Section 4).
- Comprehensive experiments to assess proposed works using industry 32-nm technology (Section 3.4 and Section 4.4).

2. LOAD CONSTRAINT OF PULSER

There is a strict upper-bound of the load that a pulser can drive to ensure the shape of the clock pulse that warrant a correct operation. If the load constraint is violated, possibly from larger wire capacitance than expected from a poor placement of latches, it may cause a distortion in the shape of clock pulse; this also affects the timing behavior of pulsed-latch circuits. For example, the value of the timing parameter (in particular clock-to-Q delay) may increase too much from its nominal value; even worse, the latch may fail to capture its input data, which leads to a malfunctioning of pulsed-latch circuits.

The load capacitance of pulser C_p for a latch group $\mathbb{G}_i$ consists of the wire capacitance C_w and the clock input capacitance of latches C_l:

$$C_p(\mathbb{G}_i) = C_w + C_l\,|\mathbb{G}_i|, \tag{1}$$

where $|\mathbb{G}_i|$ is the number of latches in $\mathbb{G}_i$. C_w is obtained by multiplying C_m with $W(\mathbb{G}_i)$; C_m is the capacitance per unit length of metal wire and $W(\mathbb{G}_i)$ corresponds to the wirelength connecting all the latches. We use a minimum spanning tree (MST) to obtain $W(\mathbb{G}_i)$ while ignoring the details of wiring such as via capacitance and different metal layers; this simplification is reasonable since using a dedicated metal layer for clock signal is a common practice.

Figure 2(a) shows an experimental setup to test the impact of over-loading a pulser. We varied the number of latches and the value of a wire capacitance to distort the shape of pulse; if data is not captured by any of the latches or the clock-to-Q delay of any latch is larger than the nominal value by 10% (due to distortion of pulse shape), we regard it as a failure. The shmoo plot of the result is shown in Figure 2(b). The value of C_p at the boundary of success and failure in Figure 2, denoted by C_{max}, turns out to be quite consistent, which allows us to approximate C_{max} as a constant in a given technology node.

3. PULSER INSERTION

Pulser insertion is performed after initial placement so that latch locations are given. Since a netlist does not include pulsers during the initial placement, the overhead of extra pulsers must be considered to avoid dramatic perturbation to the initial design; the number of pulsers can be roughly pre-calculated from the total number of latches in the netlist, which may be used to estimate the extra space for additional pulsers.

Pulsers consume a significant portion of total power, e.g. up to 50%, based on our experiments; so their number should be minimized to achieve low power. We also try to minimize the wirelength to route clock signal, i.e., $\sum_{\mathbb{G}_i \in \mathbb{G}} W(\mathbb{G}_i)$, which helps reducing clock power as well as improving routability. We now state the pulser insertion problem:

Problem 1 Given a placed design with all latch locations determined, the pulser insertion problem is to find a set of latch groups $\mathbb{G} = \{\mathbb{G}_1, \mathbb{G}_2, \cdots, \mathbb{G}_N\}$, where each $\mathbb{G}_i \in \mathbb{G}$ will be assigned a pulser, with the objective of minimizing both N and $\sum_{\mathbb{G}_i \in \mathbb{G}} W(\mathbb{G}_i)$ while $C_p(\mathbb{G}_i)$ is no greater than C_{max}.

3.1 Graph Formulation

To solve Problem 1, we introduce a weighted graph $G = (V \cup \{r\}, E)$ where each $i \in V$ is a latch and there is an edge if two nodes can be grouped. Basically G is a complete graph; every node pair i and j has an edge (i, j) with weight w_{ij} representing the capacitance of interconnect to connect them*, respectively. We introduce a dummy root node r to distinguish each latch group as a sub-tree of r in our heuristic algorithm (Section 3.2).

Problem 1 is equivalent to finding a minimum spanning tree (MST) in G while the cost of each sub-tree incident on r is no greater than C_{max}; the cost of a sub-tree is $\sum w_{ij} + nC_l$ where n is the number of nodes in the sub-tree. An example of an MST with the root r is shown in Figure 3(a). Each sub-tree of r in Figure 3(a) corresponds to a pulser group in Figure 3(b). We assume that a pulser is located on the MST as shown in Figure 3(b). Note that r is a dummy node and does not have a corresponding physical instance; therefore r and its edges are not realized in Figure 3(b). The weight of its edges w_{ri} has to be sufficiently large, e.g. $w_{max} = \sum_{\forall i,j \in V} w_{ij}$, to minimize the number of sub-trees of r in the MST. Problem 1 can be restated as follows:

Problem 2 Given a weighted graph $G = (V \cup \{r\}, E)$, find a minimum spanning tree with root r such that the number of sub-trees incident on r is minimized while the cost of each sub-tree, i.e., $\sum w_{ij} + nC_l$, is no greater than C_{max}

*w_{ij} is obtained by multiplying C_m with Manhattan distance between i and j, and thus $w_{ij} = w_{ji}$.

Algorithm *NS_Cluster*

L1 Generate graph $G = (V \cup \{r\}, E)$
L2 $V \leftarrow$ set of latches, $E \leftarrow \bigcup_{i \in V}(r, i)$
L3 **while** $\Delta_{max} > 0$ **do**
L4 $\Delta_{max} \leftarrow 0$
L5 **for** each $i \in V$ and $j \in T_n(i)$ **do**
L6 Compute $\Delta \leftarrow w_{kj} - w_{ij}$, where k is parent of j
L7 **if** $E \cup (i,j) \setminus (k,j)$ satisfy C_{max} **and** $\Delta > \Delta_{max}$ **then**
L8 $\Delta_{max} \leftarrow \Delta$, $i_{max} \leftarrow i$, $j_{max} \leftarrow j$
L9 Update $E \leftarrow E \cup (i_{max}, j_{max}) \setminus (k_{max}, j_{max})$
L10 Sub-trees of r are locally refined to form MST

Figure 4: Pseudo-code of *NS_Cluster* algorithm.

3.2 Heuristic Algorithm

A heuristic algorithm to solve Problem 2 is shown in Figure 4. We named it *NS_Cluster* since it is based on neighbor search; it iteratively searches the nodes in neighboring sub-trees that are physically close to the current one. The node that minimizes the overall cost is moved to the current sub-tree, thereby gradually approaching a reasonable solution.

We first generate a graph G with latch nodes and dummy root r; all latch nodes are initially connected to r, where each node becomes a sub-tree (L1–L2). For each $i \in V$, we search $j \in T_n(i)$ with maximum Δ; $T_n(i)$ is a set of nodes in the neighboring sub-trees of i and Δ is the amount of change in cost if j were to be detached from its parent k and attached to i, i.e., $w_{kj} - w_{ij}$; k is defined as a node that has an edge with j and has smaller number of hops from r than j does. If the cost of a new sub-tree does not violate C_{max} and its Δ is larger than the previous Δ_{max}, Δ becomes a new Δ_{max} and the corresponding i and j become i_{max} and j_{max} (L7–L8). Once the move with maximum Δ_{max} is found, the graph is updated by detaching j_{max} from its parent k_{max} and attaching it to i_{max} (L9). This process (L4–L9) is repeated until Δ_{max} becomes negative, i.e., there is no more move that can improve the cost. Finally, since some sub-trees may not be MSTs, we refine connection within each sub-tree to form an MST (L10). The complexity of the algorithm is $O(n^2)$, where n is the number of latches. The number of iterations of L5 has the bound $O(n)$ since $T_n(i)$ does not scale with the problem size and can be considered as a constant . The number of iterations of L3 also has the bound $O(n)$; it, however, converged much faster in practice.

Figure 5 shows an example of *NS_Cluster*. There are six nodes in the graph, and they are initially connected to r (Figure 5(a)); weight of edges connected to r is set to w_{max} to minimize the number of sub-trees as they are built incrementally. The bold arrows indicate node pairs with Δ_{max} at each iteration. It takes four moves to obtain the final result, shown in Figure 5(e), with two latch groups: {a,b,c} and {d,e,f}.

3.3 ILP Formulation

Problem 2 can be optimally solved by formulating it as an integer linear programming (ILP). To use a flow-based formulation, we extend edges in G to have a direction; each latch pair i and j now has two directed edges (i,j) and (j,i) where $w_{ij} = w_{ji}$. Even though ILP takes too much time to find a solution, we can use it to small circuits to assess our heuristic.

The following notations are used in the ILP formulation:

- x_{ij}: a Boolean variable which indicates the existence of a directed edge (i,j) in the solution.
- y_{ij}: a non-negative variable which specifies the flow[†] on an edge (i,j); y_{ij} becomes 0 when $x_{ij} = 0$.

[†]The flow of edge (i,j) is the total capacitance of sub-tree that branches out from j including the capacitance due to edge (i,j).

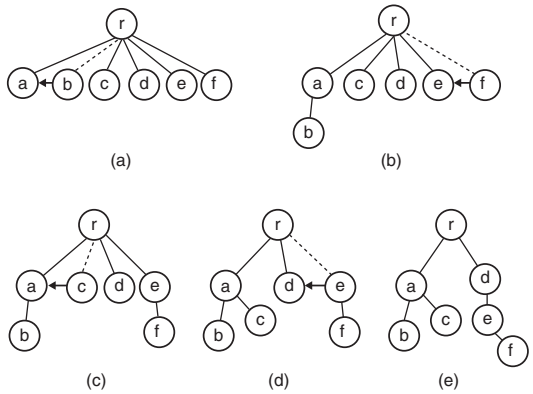

Figure 5: Example of *NS_Cluster*: (a) initially all nodes are connected to r, (b)(c)(d) a move with Δ_{max} is executed each iteration to obtain (e) final result.

Suppose that the index of r is 0 and $V = \{1, 2, \cdots, n\}$. By using a large constant for w_{0j}, the objective function in (2) naturally minimizes the number of sub-trees incident on r.

$$\text{Minimize} \quad \sum_{i=0}^{n}\sum_{j=1}^{n} w_{ij}x_{ij} \tag{2}$$

Subject to

$$\sum_{i=0}^{n} x_{ij} = 1, \qquad \forall j \in V \tag{3}$$

$$\sum_{i=0}^{n} y_{ij} - \sum_{i=1}^{n} y_{ji} = C_l + \sum_{i=1}^{n} w_{ij}x_{ij}, \qquad \forall j \in V \tag{4}$$

$$(C_l + w_{ij})x_{ij} \le y_{ij} \le C_{max}x_{ij}, \qquad \forall i, j \in V \tag{5}$$

$$C_l x_{0j} \le y_{0j} \le C_{max}x_{0j}, \qquad \forall j \in V \tag{6}$$

Constraint (3) ensures that each node has only one incoming edge. Constraint (4) is to conserve flow for each node. Constraint (5) is to set the lower and upper bounds of the flow in each edge within sub-trees; edges that are incident on r are bounded by constraint (6).

Constraints (3) and (4), together with the objective function, ensure that a solution of the ILP formulation is an MST [8]. An MST of n nodes must satisfy four conditions: 1) the sum of edge cost is minimized, 2) all nodes are connected, 3) there are $n-1$ edges, and 4) there are no cycles. The first condition is satisfied by the objective function. The second and third conditions are satisfied by constraint (3); the second condition is implied by the constraint (3) and summing the constraint over all nodes (excluding r) gives $\sum_{j=1}^{n}\sum_{i=0}^{n} x_{ij} = n-1$, which is the number of edges in a solution. From constraint (3), only possible cycles are in the form of simple cycles or simple cycles with sub-trees branching out from it; these cycles, however, cannot be formed due to constraint (4), which satisfies the last condition. For example, assume that nodes $\{1, 2, \cdots, n\}$ form a simple cycle together with edges $\{(1,2), (2,3), \cdots, (n-1,n), (n,1)\}$. If we assume that $y_{1,2}$ has non-negative value of a, then flow values of incident edges in the cycle can be obtained; we have $y_{1,2} = a - n \cdot C_l - w_{1,2} - w_{2,3} \cdots - w_{n1}$ after applying constraint (4) on all the edges in the cycle, violating our assumption that $y_{1,2} = a$. Therefore simple cycles are not pos-

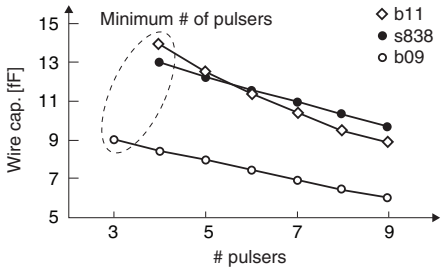

Figure 6: Trend of minimum total wire capacitance $\sum C_w$ while varying the number of pulsers N; this was obtained by including $\sum_{j=1}^{n} x_{0j} = N$ to the ILP formulation while varying N.

Table 1: Comparison between *NS_Cluster* and *Greedy* [6]. The number of pulsers (# Pulsers) and total wire capacitance ($\sum C_w$) are used to assess the quality of result

Name	# Latches	*Greedy*		*NS_Cluster*	
		# Pulsers	$\sum C_w$ (fF)	# Pulsers (%)	$\sum C_w$ (%)
s1423	74	11	33	-18.2	-10.1
s13207	230	31	87	-12.9	-1.0
s15850	442	60	178	-8.3	-3.1
s38417	1460	202	582	-9.9	-7.1
b12	119	17	50	-11.8	-8.0
b14	215	35	104	-11.4	-5.5
spi	229	31	87	-9.7	-3.7
wb_dma	522	74	214	-9.5	-6.7
pci_bridge	3267	462	1373	-10.8	-7.5
ethernet	10543	1477	4487	-10.6	-6.4
Average				-11.4	-5.8

sible solutions of the ILP formulation. Simple cycles with sub-trees cannot be formed due to similar reasoning.

3.4 Experimental Results

We carried out experiments on a set of sequential circuits taken from the ISCAS, ITC, and opencores [9] benchmarks, as shown in the first column of Table 1. Each circuit was synthesized with flip-flops by a commercial logic synthesis tool [10] using a cell library based on 32-nm CMOS technology (C_m is 0.22 fF/μm). The netlist, after converting flip-flops to latches, was then submitted to a commercial physical design tool [11] to perform initial placement; we forced 80% of the placement region to be occupied by cells. The number of latches in each circuit is reported in the second column of Table 1. A gate-level netlist together with the latch locations are given to *NS_Cluster*. We implemented *NS_Cluster* in C.

For the reference of comparison, we implemented a greedy algorithm (*Greedy*); it iteratively merges a pair of latches with the minimum distance if they do not violate C_{max} [6]. Columns 3–6 of Table 1 compares the number of pulsers and the total wire capacitance of clock pulse between *NS_Cluster* and *Greedy*. In all circuits we tested, *NS_Cluster* obtained a solution with an average of 11.4% less number of pulsers and 5.8% smaller wire capacitance than *Greedy*. This is promising result considering that using smaller number of pulsers tend to increase the total wire capacitance, as illustrated in Figure 6.

Table 2 compares the result of *NS_Cluster* with those obtained by ILP formulation and *Greedy*. Because the execution time of ILP solver [12] is substantial, we could only test circuits of small

Table 2: Comparison among ILP [12], *NS_Cluster*, and *Greedy*

Name	# Latches	ILP		*NS_Cluster*		*Greedy*	
		# Pulsers	$\sum C_w$ (fF)	# Pulsers	$\sum C_w$ (fF)	# Pulsers	$\sum C_w$ (fF)
s838	32	4	12.9	4	12.9	5	13.1
s953*	29	3	10.6	4	10.2	5	8.7
b09*	27	3	9.0	3	9.2	4	8.6
b11	30	4	13.9	5	12.8	5	14.5
ac97	65	7	21.7	8	21.4	9	21.5
aes_cipher	74	8	26.1	10	26.5	11	26.6
s1423	74	8	27.6	9	26.9	11	29.6

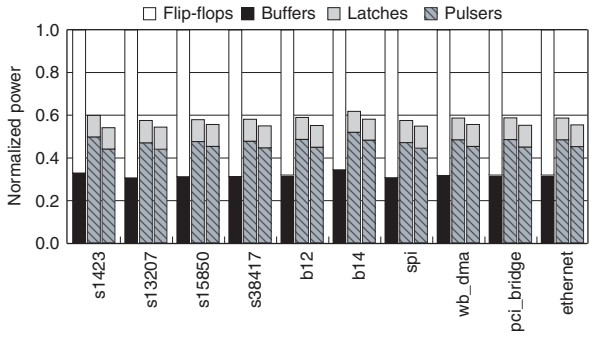

Figure 7: Comparison of normalized clocking power among flip-flips circuits (left bars), pulsed-latch circuits obtained by *Greedy* (middle bars), and *NS_Cluster* (right bars).

size; the maximum execution time and memory usage of ILP solver were set to 4 hours and 5G, respectively. Except for `s953` and `b09` (marked by *), all circuits exceeded the limit (either time or memory) and produced sub-optimal solutions. In contrast, it took less than a second for both *NS_Cluster* and *Greedy* to run each circuit in Table 2. For `s838` and `b09`, *NS_Cluster* found a solution close to that of ILP; the result of *NS_Cluster*is slightly worse than that of ILP for the other circuits, but better than that of *Greedy*.

The saving of clocking power by using pulsed-latches is quantitatively illustrated in Fig. 7. A fast SPICE simulator [13] was used to obtain the power consumption of pulsers and latches after including the wire capacitance to connect them; the input data of each latch was set to a constant during the simulation to capture the power consumption of latches due to clocking. For the power comparison to be fair, power consumption of leaf-level clock buffers were included in flip-flop circuits since pulsers can be considered as clock buffers that form leaves of the clock tree; the number of clock buffers and their size were determined such that the slew of the clock delivered to flip-flops match that of the clock pulse in pulsed-latch counterparts. On average of ten circuits, the clocking power is reduced by 41.2% and 44.7% (compared with flip-flop designs) for pulsed-latch designs obtained by *Greedy* and *NS_Cluster*, respectively. The large saving comes from replacing power-consuming flip-flops with latches; a flip-flop has more transistors, including an inverter inside a flip-flop to generate inverted clock, that are switched by clock signal even though its input data do not change. The reduction in both the number of pulsers and wire capacitance led to an average of 5.9% more saving in the clocking power of pulsed-latch designs obtained by *NS_Cluster* compared with that by *Greedy*.

Algorithm *LD_Cluster*

L1 Generate graph $G(V,E)$, $w_{uv} \leftarrow \mathrm{PD}_{uv}$, $n_{itr} \leftarrow 0$
L2 **while** $E \neq \emptyset$ **or** $n_{itr} < N_{itr}$ **do**
L3 Find pair of nodes u and v with the smallest w_{uv}
L4 **if** $|u| + |v| \leq 4$ **then**
L5 $u' \leftarrow$ merge u and v, $n_{itr} \leftarrow n_{itr} + 1$
L6 Update location and edge weights of u'
L7 **else then** Remove edge (u,v)
L8 **if** $|u'| = 4$ **then** Remove all edges connected to u'
L9 **if** $w_{uv} = \mathrm{PD}_{uv}$ **and** $w_{uv} > \mathrm{PD}_{th}$ **then**
L10 Update all $w_{uv} \leftarrow \alpha \mathrm{LD}_{uv} + (1-\alpha)\mathrm{PD}_{uv}$

Figure 8: Pseudo-code of *LD_Cluster* algorithm.

4. DESIGN OF PULSED-REGISTER CIRCUITS

A pulsed-register is a group of latches with an integrated pulser. Therefore the problem of routing clock signal from a pulser to the latches is naturally resolved. However, these benefits come at the cost of more restriction in placement. Placing multiple latches abutted to each other to form a single pulsed-register can degrade the quality of the placement solution such as timing and wirelength [14]. In our experiment, the maximum number of bits for pulsed-registers is set to four, which is also common in an industry practice, to restrain the impact of using pulsed-registers [15].

To use pulsed-registers, we first need to determine groups of flip-flops in an initial netlist designed using flip-flops. A common practice is to identify flip-flop groups based on their physical distance in an initial placement [16]; the rationale is to find a clustering solution that would produce a minimal perturbation to the initial placement. If we can identify registers that are from the same pipeline stage or constitute a vector in RTL-level description, they are good candidates for grouping [16]. This approach, however, may not be applicable in random logic designs found in most ASICs. Fortunately, the concept of hierarchy is popularly employed in recent SoC (System-on-Chip) style designs and there is significant potential to capitalize on the hierarchy information during physical optimization [17]. To extract the similar association between flip-flops, we propose a concept of logical distance to formulate a better metric that can identify flip-flop groups for pulsed-registers.

4.1 Graph Formulation

To formulate pulsed-register problem, we introduce a weighted graph $G = (V,E)$; each $v \in V$ is either a flip-flop or a set of flip-flops and there is an edge $(u,v) \in E$ if the physical distance between u and v, i.e., $|x_u - x_v| + |y_u - y_v|$[‡], is smaller than D_{max}. The value of D_{max} is empirically determined so that two flip-flops placed far from each other do not have an edge in G. Each edge has a weight w_{uv} that models the cost of merging u and v; its value is initially set to PD_{uv}, which is a normalized physical distance between u and v, i.e., $\frac{(|x_u - x_v| + |y_u - y_v|)}{D_{max}}$.

We assume that a maximum number of iterations N_{itr} is given to control the extent of merging (L2 in Figure 8). The pulsed-register problem is formulated as a problem of merging nodes in G:

Problem 3 Given a graph $G = (V,E)$ and N_{itr}, merge nodes in G (where each merged node can have up to 4 nodes) to obtain G' such that the impact on physical design is minimized when all nodes in G' are converted to pulsed-registers.

[‡](x_u, y_u) and (x_v, y_v) are x- and y-coordinates of u and v, respectively, in the initial placement.

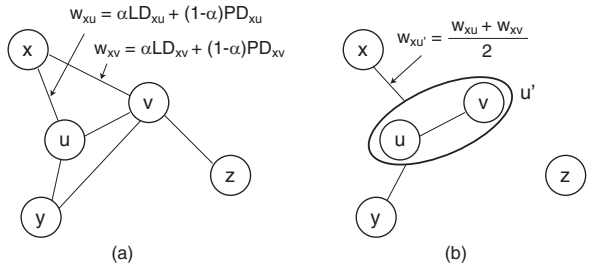

Figure 9: (a) Before merging and (b) after merging u and v.

To quantify the impact on physical design, the wirelength (both signal and clock) and the delay of the critical path are measured, as shown in Table 3.

4.2 Logical Distance

If a pair of flip-flops are from the same hierarchical block, it is more likely that they are tightly connected to each other than other flip-flops from different hierarchical blocks. We introduce a logical distance to measure the logical connectivity between flip-flops using a logic depth (or level) to their common gates. We first assign levels $l_{f,v}$ and $l_{b,v}$ to each gate v, which are given by

$$l_{f,v} = \max_{u \in FI(v)} l_{f,u} + 1, \quad l_{b,v} = \max_{u \in FO(v)} l_{b,u} + 1, \tag{7}$$

where $FI(v)$ is the set of gates in the fan-in cone of v and $FO(v)$ is the set of gates in the fan-out cone of v. To obtain $l_{f,v}$ and $l_{b,v}$, levelization is performed in both forward (direction of data propagation) and backward (opposite direction of data propagation) directions, respectively, and it stops if the level of a gate reaches a maximum level L_{max}; small value of L_{max}, e.g., 5, is good enough for capturing the logical distance.

Then the logical distance between flip-flops u and v is defined by

$$\mathrm{LD}_{uv} = \frac{1}{2L_{max}^2}\Big(\min_{x \in FI(u) \cap FI(v)} l_{b,x}^2 + \min_{x \in FO(u) \cap FO(v)} l_{f,x}^2\Big). \tag{8}$$

If $FI(u) \cap FI(v)$ or $FO(u) \cap FO(v)$ is an empty set, the corresponding minimum level is set to L_{max}. Note that LD_{uv} takes a normalized value with its maximum up to 1. We noticed that various forms of LD_{uv} produced the similar result as long as the concept of logical distance exits in the metric.

4.3 Clustering Algorithm

Figure 8 shows the pseudo-code of our clustering algorithm to solve problem 3; it is a greedy algorithm that iteratively selects a node pair in G to gradually grow each latch group. Initially, each node represents a flip-flop and w_{uv} is set to PD_{uv} (L1). For each iteration, a node pair u and v with smallest w_{uv} is found (L3). Merging u and v is valid if the cardinality of their merged node does not exceed 4 (L4). Note that each node could imply more than one flip-flop after merging. If the merging is valid, then two nodes are merged into a new node u' (L5–L6); the location of u' is the median of u and v, and the weight of edges connected to u' are updated by taking the average of weights connected to u and v. An example of merging u and v is shown in Figure 9. The nodes that have edges with both u and v now have an edge with merged node denoted by u' in Figure 9(b). The edge between z and v is removed after merging because there is no edge between u and z. The weight of edges connected to u' is also updated by taking the average of edge weights before merging. If u and v cannot be merged, the edge between them is removed (L7). If the cardinality of u' is 4, all edges

Table 3: Comparison of result from initial flip-flop designs (Initial), pulsed-register designs obtained by considering only physical distance (*PD_Cluster*) and both logical and physical distances (*LD_Cluster*). The results of *PD_Cluster* and *LD_Cluster* are expressed as the percentage of change from the initial flip-flop designs

Name	# Latches	Initial			*PD_Cluster*			*LD_Cluster*		
		Signal (μm)	Clock (μm)	Delay (ns)	ΔSignal (%)	ΔClock (%)	ΔDelay (%)	ΔSignal (%)	ΔClock (%)	ΔDelay (%)
s1423	74	1910	232	1.28	10.1	-12.4	0.0	5.2	-13.4	3.1
s13207	230	5905	666	0.61	7.3	-2.7	0.0	7.4	-8.5	0.0
s15850	442	16105	1474	1.28	8.2	-16.7	-1.6	7.9	-16.4	0.8
s38417	1460	40372	4220	1.85	10.1	-3.7	3.8	9.9	-2.5	3.2
b12	119	3843	399	0.94	15.2	-6.6	2.1	7.9	-11.7	-1.1
b14	215	38404	1074	3.15	2.2	-8.8	0.0	1.8	-9.8	-0.3
spi	229	18659	727	2.20	4.0	-2.5	-0.9	3.9	-5.0	-0.9
wb_dma	522	33005	1615	1.55	5.4	-8.1	3.2	4.9	-4.9	-1.3
pci_bridge	3267	121574	10494	2.19	7.8	-8.0	0.0	7.5	-6.2	1.8
ethernet	10543	482217	34536	3.55	6.4	-8.0	-3.7	6.1	-6.8	-2.8
Average					7.7	-7.8	0.3	6.3	-8.5	0.3

connected to u' are removed since it cannot be merged with other nodes anymore (L8). This process is repeated until there is no more edges in the graph or the number of iterations n_{itr} has reached N_{itr} (L2).

After several iterations of merging, if the minimum w_{uv} in G becomes larger than some threshold PD_{th}, the logical distance comes into play (L9–L10), and w_{uv} is redefined by

$$w_{uv} = \alpha \mathrm{LD}_{uv} + (1-\alpha)\,\mathrm{PD}_{uv}, \tag{9}$$

where α is empirically determined for each circuit; we incremented α by 0.05 (from 0.05 to 1.0) and selected the one that gives the minimum signal wirelength. Typically, α ranges from 0.05 to 0.4, The rationale of changing w_{uv} is that when PD_{uv} is small, i.e., $\mathrm{PD}_{uv} \leq \mathrm{PD}_{th}$, it is a good metric for finding flip-flops for merging. However, if it becomes large, i.e., $\mathrm{PD}_{uv} > \mathrm{PD}_{th}$, it implies that distant flip-flops are clustered together, and from this point, considering LD_{uv} along with PD_{uv} is more effective in finding flip-flops for merging.

4.4 Experimental Results

We carried out experiments on the same benchmark circuits in Section 3.4: the same setting and technology were used to obtain the netlist and the initial placement of flip-flop circuits. The flop-flop locations from the initial placement become input to our clustering algorithm, i.e., *LD_Cluster*, implemented in C. N_{itr} was obtained for each design by multiplying 0.6 to the number of flip-flops. Finally, flip-flop groups obtained by *LD_Cluster* are converted to pulsed-registers to obtain pulsed-register circuits followed by legalization, clock tree synthesis, and incremental placement and routing [11].

Different bits of pulsed-registers can be created as separate cells and added to the library, or can be formed by abutting latches and pulsers via running a Tcl script from a commercial physical tool [11]; we used the latter approach to avoid designing a new cell for the sake of experiment. An example of forming a pulsed-register from multiple flip-flops is shown in Figure 10. The pulsed-register is placed at the median location of corresponding flip-flops in the initial placement followed by legalization; the relative location of latches within pulsed-register is determined by the relative location of corresponding flip-flops, as shown in Figure 10.

For the reference of comparison, we implemented an algorithm called *PD_Cluster*; it is the algorithm in Figure 8 without L9–L10 so that $w_{uv} = \mathrm{PD}_{uv}$ all the time. The results of initial flip-flop designs and those obtained by *LD_Cluster* and *PD_Cluster* are compared in Table 3; the results were obtained by using a commercial tool [11]. Columns 3–5 of Table 3 show the wirelength of signal (Signal), the wirelength of clock (Clock), and the delay of critical paths (Delay) of initial flip-flop designs. Columns 6–8 and columns 9-11 are the results of pulsed-register designs obtained by *PD_Cluster* and *LD_Cluster*, respectively; they are expressed as the percentage of change from the result of initial flip-flop designs. On average, the signal wirelength obtained by *LD_Cluster* is increased by 6.3% from that of the initial flip-flop circuits, which is 1.4% smaller than that obtained by *PD_Cluster*; especially, the difference was 7.3% for `b12`. The clock routing wirelength obtained by *LD_Cluster* is reduced by 8.5% from that of flip-flop circuits; this is 0.7% smaller than that obtained by *PD_Cluster*.

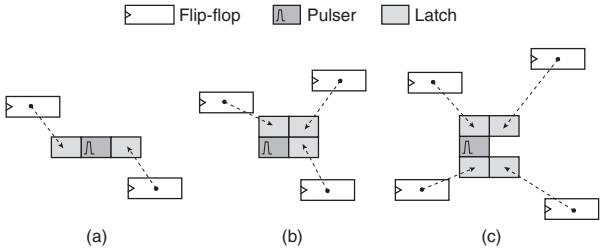

Figure 10: Converting flip-flops to abutted latches and pulser to form (a) 2-bit, (b) 3-bit, and (c) 4-bit pulsed-registers.

In terms of delay and power, the average result between the two approaches are about the same although they vary from circuit to circuit. The delay of critical paths, which was obtained after performing timing optimization [11] such as buffer insertion, increases by only 0.3% on average; their impact on clock period can be ignored when we consider the small amount of time-borrowing of pulsed-registers. Figure 11 compares the clocking power of pulsed-register circuits normalized to that of flip-flop circuits; the results were obtained using a commercial tool [11]. The clocking power is reduced by 23.9% and 24% (compared with flip-flop designs) on average with *PD_Cluster* and *LD_Cluster*, respectively; the saving attributes to power-efficient pulsed-registers (-12%), reduced wirelength and buffers of the clock network (-48%). A pulsed-register embed multiple flip-flops together reducing the demand of clock routing wires and buffers significantly.

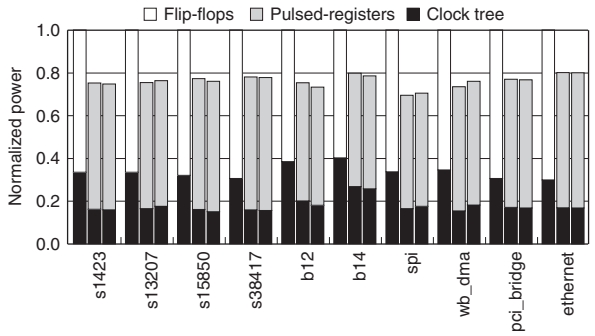

Figure 11: Comparison of normalized clocking power among flip-flips circuits (left bars), pulsed-register circuits obtained by *PD_Cluster* (middle bars), and *LD_Cluster* (right bars).

5. CONCLUSION

We have presented clustering algorithms to implement pulsed-latch and pulsed-register circuits for reducing the clocking power of conventional flip-flop circuits. To ensure the shape of clock pulse in pulsed-latch circuits, the pulser insertion problem is formulated on a given placement solution of a design to find a set of latch clusters subject to maximum capacitance constraint. To further improve the integrity of clock signals to latches, pulsed-registers have been used with logical-distance based clustering enhancement.

To make pulsed-latch and pulsed-register circuits robust to noise and process variations, their effect on pulsers should be considered; this is left for future work.

References

[1] R. S. Shelar, "An efficient clustering algorithm for low power clock tree synthesis," in *Proc. Int. Symp. on Physical Design*, Mar. 2007, pp. 181–188.

[2] S. Shibatani and A. Li, "Pulse-latch approach reduces dynamic power," July 2006, EE Times.

[3] H. Li, M. Chen, and K. Ho, "System and method of replacing flip-flops with pulsed latches in circuit designs," U.S. Patent 7694242 B1, April 2010.

[4] Y. Shin and S. Paik, "Pulsed-latch circuits: a new dimension in ASIC design," *IEEE Design & Test of Computers*, 2011, accepted for publication.

[5] S. Kim et al., "Pulser gating: A clock gating of pulsed-latch circuits," in *Proc. ASPDAC*, Jan. 2011, pp. 190–195.

[6] Y. Chuang et al., "Pulsed-latch-aware placement for timing-integrity optimization," in *Proc. DAC*, June 2010, pp. 280–285.

[7] H. Partovi et al., "Flow-through latch and edge-triggered flip-flop hybrid elements," in *Proc. ISSCC*, Feb. 1996, pp. 138–139.

[8] B. Gavish, "Formulations and algorithms for the capacitated minimal directed tree problem," *JACM*, vol. 30, no. 1, pp. 118–132, Jan. 1983.

[9] "OpenCores," http://www.opencores.org/.

[10] Synopsys, "Design Compiler User Guide," June 2009.

[11] Synopsys, "IC Compiler User Guide," June 2009.

[12] IBM, "IBM ILOG CPLEX v12.2," 2009.

[13] Synopsys, "NanoSim User Guide," June 2009.

[14] Y. Cheon et al., "Power-aware placement," in *Proc. DAC*, June 2005, pp. 795–800.

[15] Y. Chang et al., "Post-placement power optimization with multi-bit flip-flops," in *Proc. ICCAD*, Nov. 2010, pp. 218–223.

[16] W. Hou, D. Liu, and P.-H. Ho, "Automatic register banking for low-power clock trees," in *Proc. ISQED*, Mar. 2009, pp. 647–652.

[17] Y. Chuang et al., "Design-hierarchy aware mixed-size placement for routability optimization," in *Proc. ICCAD*, Nov. 2010, pp. 663–668.