

# Tolerating Process Variations in High-Level Synthesis Using Transparent Latches

Yibo Chen and Yuan Xie

Department of Computer Science and Engineering  
The Pennsylvania State University, University Park, PA 16802  
Email: {yxc236, yuanxie}@cse.psu.edu

**Abstract**—Considering process variability at the behavior synthesis level is necessary, because it makes some instances of function units slower and others faster, resulting in unbalanced control steps and reducing the attainable frequency of the circuit. To tackle this problem, this paper proposes a methodology to replace the edge-triggered flip-flops by transparent latches, to exploit latches' extra ability of passing time slacks and tolerating delay variations. In the paper we first define the timing yield in high-level synthesis, and then present how to replace flip-flops with latches to improve timing yield and mitigate the impact of process variations. We then discuss the benefits and overheads for the replacement, and propose an optimization framework for latch replacement in high-level synthesis design flow. Experimental results show that the latch-based design can achieve an average of 27% improvement of timing yield compared with traditional flip-flop based design.

## I. INTRODUCTION

Process variation has become one of the most critical design challenges in nanometer VLSI era. The scaling down of technology has resulted in significant deviations from the nominal values of transistor parameters, such as channel length, threshold voltage, and metal linewidth. The magnitude of variations in gate length, as an example, increases from 35% in a 130 nm technology to almost 60% in a 65 nm technology [1], resulting in the large delay variations in circuit units.

In the past few years, the process variation problem has been addressed at all levels of circuit design, from high level abstraction to physical level implementation [2]–[4], wherein high-level synthesis (HLS) has attracted very active attention and a lot of recent work tries to cope with the variation problem at the very early stage of a design process. HLS mainly consists of three steps: scheduling, resource sharing, and resource binding. Traditionally, all these steps are performed based on deterministic analysis. In the era of statistical timing analysis, using a deterministic value for the delay of function units may not be appropriate any more, and binding an operation between fixed clock cycles to a function unit with statistical delay distribution will naturally result in performance yield loss. Compared with the requirement of 100% performance yield in worst-case based HLS, the basic objective of variation-aware HLS is to maximize performance yield while minimizing overhead on area or latency [5], [6].

Traditional HLS assumes edge-triggered flip-flops as storage elements, and the use of transparent latches has not been well studied. In fact, compared with flip-flops, latches have many merits in power, area (usually latches have a half of the size of flip-flops), and especially in tolerating process variations, since latches are transparent

during the active clock period and have the extra capability to pass time slacks between control steps. Using latches in HLS may lead to significant improvement of performance yield, since slower function units may have more time slack to finish execution and therefore the probability of satisfying the timing requirement is increased. However, designs using latches are prone to hold-time violation, because it is difficult for latches to hold the output value when input switches during the active clock period. Usually two-phase non-overlapping clock is required to fix this problem, which in turn increases the design complexity [7].

This paper proposes a new methodology to replace flip-flops with latches as storage elements in high-level synthesis, while preserving the latch's extra capability to tolerate process variations and meanwhile ensuring that no timing violation occurs. In the proposed methodology, we first introduce the variation-aware HLS, emphasizing the capability of tolerating process variations in latch based design compared with flip-flop based design. Then we present how to replace flip-flops with latches to avoid the timing violations, and also discuss the required modification on resource sharing in latch-based design. Experiments are conducted on several HLS benchmarks to show the effectiveness of performance yield improvement in latch based design with minimal area overhead.

## II. RELATED WORK

Prior work tightly related to this paper mainly falls into three categories: circuit optimization based on statistical timing analysis, timing variation-aware HLS, and latch based circuit design.

Recently both academia and industry have paid great attention to statistical timing analysis as process variability becomes a critical problem in modern VLSI design. Various optimization techniques have been proposed based on the results of statistical analysis, such as buffer insertion, gate sizing, and threshold voltage assignment [1]. However, most of these optimizations focus on device level or gate level. It has been demonstrated that to efficiently reduce the impact of process variation, it's better to incorporate variability at higher level of circuit design [8].

Some recent work attempted to consider process variation during high-level synthesis. Hung et al. [5] proposed a *simulated-annealing* based high-level synthesis framework to take process variations into account. Jung et al. [6] made use of statistical timing information to perform high-level synthesis based on heuristics, and implicitly schedules groups of operations into multiple clock cycles and slower function units can utilize the time slacks of faster units in these groups.

The benefit of latch based design has been studied [9]. Wu et al. [10] and Yang et al. [11] proposed approaches for power and area

This work was supported in part by NSF grants of CAREER 0643902, CNS 0720659, CCF 0702617, and a grant from SRC.

optimization of circuits by replacing some flip-flops with latches. Their algorithms are to find out flip-flops that can be replaced while preserving the functionality of the circuit, so that the power consumption is reduced. Chinnery et al. [7] presented an approach to replace flip-flops by latches automatically in an ASIC gate-level netlist for the Xtensa microprocessor. The algorithm equivalently transforms every flip-flop to a combination of two latches that are active on opposite clock phases, to provide some degree of immunity to clock skew and thus improve circuit performance.

To the best of our knowledge, it has not been attempted to incorporate process variation consideration into a latch based high-level synthesis. Therefore, the main contribution of this paper can be summarized as follows:

- It introduces a new methodology for replacing flip-flops by latches as storage elements in high-level synthesis to improve timing yield;
- It discusses the modification on register allocation and resource sharing in latch based design to eliminate timing violations;
- It constructs a framework for optimizing timing yield in latch-based design with minimized area overhead.

### III. PRELIMINARIES AND MOTIVATING EXAMPLE

In this section, we mainly introduce some preliminaries on variation-aware high-level synthesis, and discuss the motivation of replacing flip-flops with latches in high-level synthesis.

#### A. Timing Yield in High-Level Synthesis

High-level synthesis (HLS) is the process of translating a behavioral description into a register level structure description. Scheduling and resource binding are key steps during the synthesis process. The scheduler is in charge of sequencing the operations of a control/data flow graph (CDFG) in correct order and it tries to schedule as many operations as possible in the same control step to extract more parallelism. The binding process binds operations to hardware units in the resource library to complete the mapping from abstracted descriptions of circuits into practical designs.

The resource library consists of hardware units with different delay and area properties. Traditionally the worst-case latency of each function unit is provided to HLS algorithms. However, as the magnitude of process variation grows rapidly, worst-case based analysis and optimization are no longer acceptable since they introduce too much pessimism in design, and lead to greatly increased design effort to meet the requirement. Instead, statistical description and analysis of function unit delay are introduced to tackle the timing problem in high-level synthesis.

With the existence of process variations, the delay of a function unit is no longer a fixed value, but spreads into wide distributions. In a statistical timing view, the distributions can be described by a probability density function (PDF). **Timing yield** is then defined as the probability that a function unit can finish execution in a given time period, that is, the cumulated probability under a given  $T_{clk}$  in the PDF. Fig. 1(a) gives a real example of delay variations and Fig. 1(b) demonstrates the concept of timing yield. Empirical data shows that the delay of resource units approximately conforms to Gaussian distribution [5].

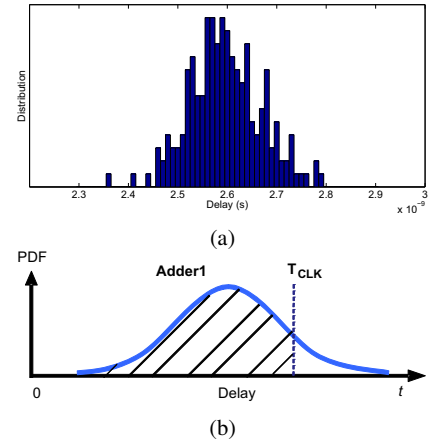


Fig. 1. (a).The delay variations of a carry-ripple adder implemented in 90nm technology; (b) Timing yield of a function unit with clock period  $t_{clk}$  in variation-aware HLS

The **overall timing yield** of a synthesized CDFG is the probability that the entire design can finish execution in given clocks steps without any timing violation. Improving the overall timing yield is the basic objective of variation-aware HLS and the key difference from worst-case analysis.

To compute the overall timing yield, we first try to decompose the CDFG. In traditional design, function units are bound to control steps separated by flip-flops. As flip-flops only latch values at the edge of the clock, the delay distributions of function units are “truncated” by the flip-flops, and no time slacks can be passed through.

Therefore, in traditional design, the delay distributions of function units are independent to each other (To simplify the discussion, at this moment we don’t consider chained operation and resource sharing yet). In statistical view, the overall timing yield is the product of the timing yields of all function units, as shown in Expression (1).

$$TimingYield_{Overall} = \prod_{i=1}^N TimingYield(i) \quad (1)$$

where  $i$  and  $N$  are the index and the total number of function units respectively. We can then infer that, the overall timing yield decreases significantly as the number of function units increase in larger design. An intuitive thinking for yield improvement is to remove the clock “barrier” and combine faster and slower units together. If slower units can borrow time slack from faster units, the probability of timing violation will consequently be reduced and the timing yield is then improved.

Putting multiple chained operations in a single cycle is a well studied technique in high-level synthesis [12], as shown in Fig. 2(b). Later we will show in detail that timing yield can be largely improved with such modifications. However, the drawbacks are also obvious: (1) As operations are combined into groups, the lifetime of each operation is prolonged by several times. Therefore it becomes harder to share resources between operations in the same type, resulting in significant area overhead. (2) The operations in the middle of the grouped chain can’t be synchronized by clocks, and thus design complexity is increased. (3) Clock cycle time is prolonged by times, so clock frequency is greatly reduced, resulting in potential difficulties in interfacing and I/O.

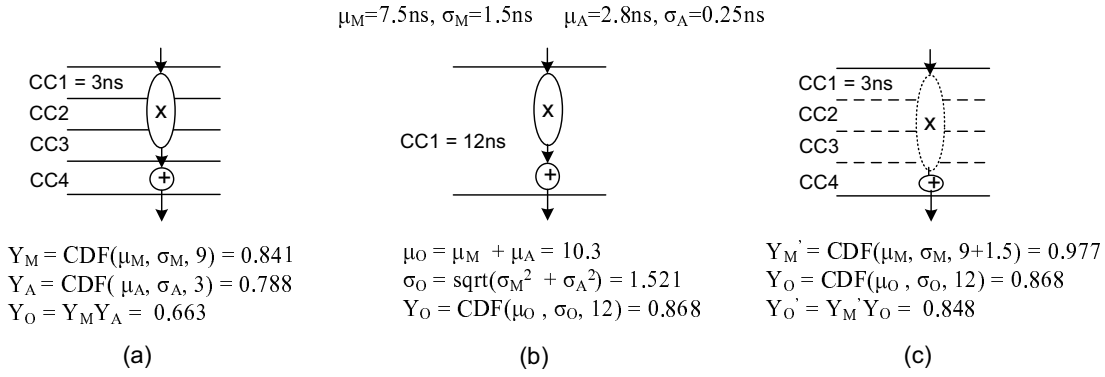


Fig. 2. Comparison of timing yield between (a) traditional design, (b) multiple chained operation and (c) latch based design.

### B. Motivation of Replacing Flip-flops with Latches

As we know, latches are transparent during the active period of a clock cycle. If control steps are separated by latches, time slacks may be easily passed between adjacent control steps, as shown in Fig. 3.  $i, j$  are two consecutive control steps in a HLS design, and

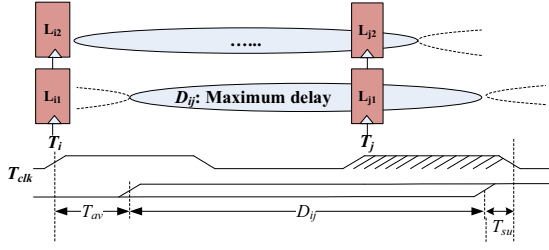


Fig. 3. Time slack passing in latch-based design.

$D_{ij}$  is the maximum delay of a function unit between step  $i$  and  $j$ .  $T_{av}$  stands for the time when data is available in step  $i$ , and  $T_{su}$  is the setup time in step  $j$ . With the active transparent characteristic of latches, we have:

$$T_{av} + D_{ij} + T_{su} \leq T_{clk} + \frac{T_{clk}}{2} \quad (2)$$

In Expression (2), if  $T_{av}$  and  $T_{su}$  are small enough and can be omitted,  $D_{ij}$  will approximately equal to  $\frac{3}{2}T_{clk}$ . That means, a slower function unit may “steal” extra execution time up to  $\frac{1}{2}$  clock cycle from adjacent control steps. Therefore, if we replace the flip-flop between a slower unit and a faster unit with transparent latch, the timing yield of these two units can be improved.

We can explain the motivation more clearly with a detailed example. In Fig. 2, an addition operation is chained after a multiplication in three difference design styles. The delay of the adder unit conforms to a Gaussian distribution with  $\mu_A = 2.8ns$  and  $\sigma_A = 0.25ns$ , while the delay of the multiplier conforms to a distribution with  $\mu_M = 7.5ns$  and  $\sigma_M = 1.5ns$ . The clock cycle for (a) and (c) is set as  $3ns$ , and the timing yield of each operation is calculated as the CDF function at different time points. In (a), the traditional flip-flop based design, at most one operation is scheduled into a clock cycle. So the overall timing yield is the product of the timing yields of the two operations, that is,  $0.841 \times 0.788 = 0.663$ . In (b), two operations are scheduled into one clock cycle, and the delay of these grouped operations approximately conforms to a new Gaussian distribution [13] with  $\mu = \mu_M + \mu_A = 10.5ns$  and

$\sigma = \sqrt{\sigma_M^2 + \sigma_A^2} = 1.52ns$ . For comparison we set the clock cycle time in (b) as  $12ns$ , yielding the same overall latency as the other two design styles. The overall timing yield in (b) is then calculated as the CDF function of the delay of the grouped operation at time point  $12ns$ , which is  $0.868$ . In (c) latches are used as the storage element between these two operations, which can then also be dealt with as grouped operations. However, since the maximum time slack that latches can pass is  $\frac{1}{2}$  clock cycle, we have to exclude the probability that the adder’s delay exceeds  $3\frac{1}{2}$  clock cycles. The CDF function of the multiplier at  $3\frac{1}{2}$  clock cycles is  $0.977$ , so the overall timing yield of (c) is  $0.868 \times 0.977 = 0.848$ .

The motivating example shows that both operations chaining and latch-based design can greatly improve the timing yield. It should be mentioned that, only one storage element in Fig. 2(c) is replaced by latch. If more latches are used in the design, we can intuitively infer that more timing yield improvement can be obtained.

As mentioned early in this section, operation chaining may lead to large area overhead. Then, what’s the side effect if we replace flip-flops with latches? In the next section, we will mainly discuss the benefits and overhead of latch-based design in general settings.

## IV. CONVERTING TRADITIONAL DESIGN TO LATCH-BASED DESIGN

In this section, we will discuss how to convert traditional design to latch-based design, and present a modification scheme for eliminating the potential hold time violations and resource sharing violations brought by latch replacement.

### A. Potential Violations in Latch-based Design

While dealing with latch-related circuits, we have to keep in mind that latches can not hold the output value when input switches during the active clock period. If flip-flops are directly replaced by latches, it’s possible for the circuits to have hold time violations. As shown in Fig. 4(a), in the first cycle, the register **Rs** holds variable **a**, and function unit **FU** evaluates **a**. At the next cycle, the register **Rs** holds a new value **b** and at the same time the function unit begins to evaluate **b**. Therefore it’s hard for register **Rx** to latch the output value **FU(a)** because of the transparent characteristic of latches.

To safely implement a storage element with a latch in such a situation, the condition is that it should not change its value until one more extra cycle after its death time, as shown in Fig. 4(b). The output of the function unit doesn’t change while register **Rx** is

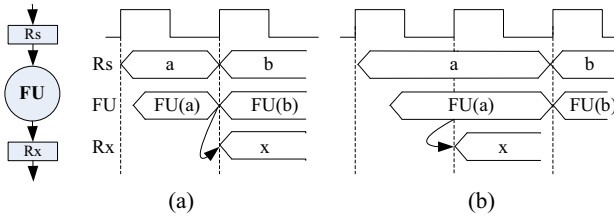


Fig. 4. Hold time violation of storage elements using latches

(a) Flip-flop Design A (b) Latch Design A (c) Latch Design B

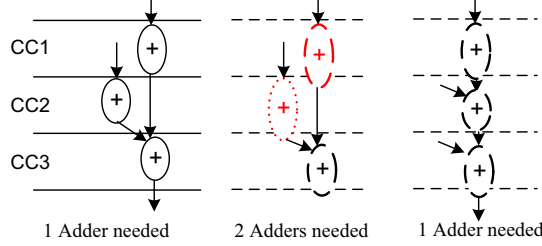


Fig. 5. Potential violation on resource sharing in latch based design

trying to latch the output value  $\mathbf{FU(a)}$ . In other words,  $\mathbf{FU}$  cannot be immediately “reused” in latch-based design. The word “reuse” here indicates the situation that for two consecutive operations, the second one takes a new input other than the output of the first one. As a matter of fact, if the second one takes the output of the first one (for instance, in the second step  $\mathbf{FU}$  doesn’t evaluate  $\mathbf{b}$ , but evaluates  $\mathbf{Rx} = \mathbf{FU(a)}$ ), no intermediate result has to be exported and consequently the potential hold time violation will not happen. In such situation, no extra modification is required for the latch replacement.

The characteristic of latches requires the life time of some function units to be extended. As a result, the resource sharing in latch based design will then be affected. In traditional flip-flop based design, function units can be reused in two consecutive control steps. As shown in Fig. 5(a), in traditional design A, all three operations can share the same function unit, so only one adder is needed in this example; however, in Fig. 5(b) for the latch-based design A, the operation in CC2 can not share the same function unit with the operation in CC1, as the latch between CC1 and CC2 cannot store the evaluation result of CC1 stably when it tries to take a new input at CC2. Therefore the operation in CC2 should be assigned to a new instance of function unit. The operation in CC3 can still share the same unit with the operation in CC1, so totally two adders are needed for the latch based design A. In Fig. 5(c), a latch based design B with different input/output relations is shown for comparison. While all operations are chained in design B, no intermediate results need to be latched. The resource sharing is therefore not affected and only one adder is needed in the design.

### B. Latch Replacement with Lifetime Extension

In order to safely replace flip-flops with latches, a modification in lifetime of some function units is required during the high-level synthesis process. We first define a term called **critical operation**. *If two operations in consecutive control steps are assigned to the same function unit in traditional HLS, and the second operation does not take the output of the first operation as input, the first one is referred as an **critical operation**.* For instance, the operations in steps CC1

and CC2 in Fig. 5(b) are critical operations.

As we discussed earlier in this section, in order to eliminate the timing and resource sharing violations in latch based design, the life time of these critical operations should be extended by one extra cycle. Given  $V_{critical}$  as the set of critical operations,  $t_d(v_i)$  and  $t_r(v_i)$  as the death time and renewing time of the function unit  $v_i$ , respectively, we have:

$$t_r(v_i) = t_d(v_i) + 1, \quad for \quad \forall v_i \in V_{critical} \quad (3)$$

We can perform the modification during the resource binding and sharing stage in the high-level synthesis process. With the modification on life time of function units, the replacement for transparent latches is guaranteed with no timing and resource sharing violations.

## V. LATCH REPLACEMENT FRAMEWORK

In this section, we will briefly discuss the latch replacement framework, evaluate the timing yield improvement and estimate the area overhead after replacement.

### A. Data Preparation

In our proposed framework, latch replacement is done during the register allocation stage after scheduling and resource binding in a HLS design flow. The input of our framework is a synthesized CDFG with bound function units.

The given CDFG is synthesized by a traditional ASAP scheduler [14]. However, as the delays of function units are statistical distributions, they can not be processed by a traditional scheduler. Therefore, we have to decide a fixed characteristic delay for each function unit, in order to perform the scheduling and resource sharing. As worst-case analysis is proved to be too pessimistic [1], the worst case value of the delay distribution is not applicable. Also, the mean value of delay distribution is not appropriate with a low timing yield of 50% for single cycle operations. In this paper, we adopt the  $1-\sigma$  sample, that is, the  $(\mu + \sigma)$  value of the delay distributions, as the characteristic delay of each function unit. For instance, given a multiplier with mean delay  $\mu = 7.5ns$  and variation  $\sigma = 1.5ns$ , the characteristic delay for this multiplier is  $7.5 + 1.5 = 9ns$ .

With the characterized delay values of function units, the scheduling and resource binding are performed in existing HLS design flow, resulting in synthesized CDFGs.

### B. Replacement Algorithm

Fig. 6 shows the flow of our proposed replace framework. The latch replacement algorithm takes the synthesized CDFG as input, finds out critical operations, modifies their life time, and assigns resources according to the modified life time.

The critical operations can be identified via a full search, as shown in Fig. 7. An initial resource binding and sharing step is performed on the input CDFG prior to our framework. Therefore, to tell whether an operation A is critical, we only need to scan the operations scheduled in the next step after A’s finish time, and check whether they share the same function unit with A. If no sharing occurs, or the sharing occurs between two chained operations (the next operation takes A’s output as input), A is then a non-critical operation.

As discussed in Section IV, the life time of critical operations should be extended by one extra cycle. The modification can be easily

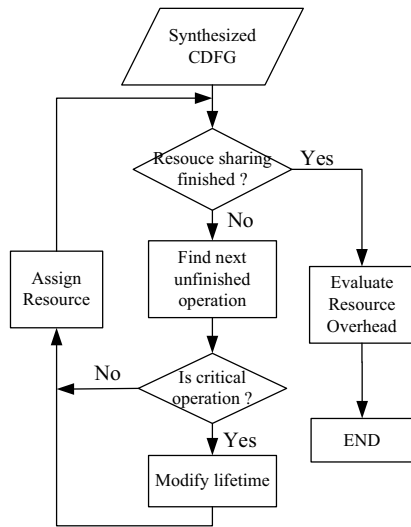


Fig. 6. Flowchart of the latch replacement framework

**Algorithm for identifying critical operations****Function:** *isCritical(OP, CDFG)***Input:** *pre-bound and resource-shared CDFG***Input:** *given operation OP***Output:** *whether the operation is critical***begin***find out the next step S after the finish time of OP;**critical = false;***for** each operation *k* start on *S***if** (*k.type == OP.type* && *k.resID == OP.resID*&& *OP ≠ k.predecessor*)*critical = true;**break;***end if****end for***return critical;***end**

Fig. 7. Algorithm outline for identifying critical operations.

performed inside a greedy search algorithm for resource re-sharing, as shown in Fig. 8. An as-soon-as-possible (ASAP) strategy is used to determine whether an operation should share the function unit with previous operations and which instance of function unit it will share.

**C. Evaluation of Timing Yield Improvement**

In Section III, Fig. 2(c), the overall timing yield of the sample latch design is calculated using conditional probability. However, after the latch replacement on the full CDFG, operations are no longer separated by register “barriers”. Instead, they are tightly coupled by the slack borrowing in timing. With more operations with multiple input/output are involved, it’s difficult to get an analytical expression of the overall timing yield. Therefore, existing analytical approaches [13] for statistical timing analysis are not applicable in latch based design. We then adopt the Monte Carlo analysis [15] for the overall timing yield computation in latch based design.

To emulate process variations, joint normal distributions on delays of function units are constructed using predefined parameters. For each run of the Monte Carlo analysis, a set of samples of the normal distributions is generated. Then, a full timing check against Expression (2) in Section III is performed on the processed CDFG with the sample delay of function units. If no timing violation is found, the run is marked as a successful one. After all runs are finished (in our experiments 100000 runs are performed in a round of

**Modified resource sharing algorithm****Input:** *synthesised CDFG***Output:** *the quantity of required resources***begin***given synthesised CDFG;**determine the quantity of required resources by using ASAP;***for** each operation *k* in CDFG*decide the resource type *i* for operation *k*;**done = false;***for** (each resource track *t* in type *i*);**if** (*t.last is critical* && *k.start\_time* >= *t.finish\_time* + 1|| *!(t.last is critical)* && *k.start\_time* >= *t.finish\_time*)*enqueue k into t;**t.finish\_time = k.finish\_time;**done = true;***end if****end for****if** (*done = false*)*allocate a new resource track *tnew* in type *i*;**enqueue k into *tnew*;**tnew.finish\_time = k.finish\_time;***end if****end for***output the number of resource tracks for each type;***end**

Fig. 8. Outline of the modified resource sharing algorithm.

analysis), the ratio of successful runs is adopted as the overall timing yield of the CDFG after latch replacement.

**D. Overhead Estimation**

The evaluation results of the aforementioned algorithms in this section show that, the modification by latch replacement will consequently yield an increase in the number of required resources.

We can then estimate the overall area overhead and compare it against the operation chaining based flip-flop design. It has to be mentioned that, less gates are needed to implement latches than those of flip-flops (typically a half of gate count for latches), so when we estimate area overhead for latch based design, the area “saving” of latches over flip-flops should be accounted for. With such area savings, the area overhead brought by increased function units can be largely compensated, yielding a relatively small overhead for the whole design, as we will see in the experimental results section.

**VI. EXPERIMENTAL RESULTS**

In this section, we present the experimental results of our latch replacement framework for high-level synthesis. The results show that our method can effectively improve the overall timing yield of given designs and reduce the impact of process variations.

We implement our latch replacement algorithm in C++ and conduct the experiment on a set of high-level synthesis benchmarks: a differential equation solver (DES), an FIR filter (FIR), a 16-point elliptic wave filter (EWR), an autoregressive lattice filter (AR) and an IIR filter used in the industry (CHEM) [5]. For the function units used in the experiments, we run Monte Carlo simulations in HSPICE to obtain the delay distributions and build a resource library with delay distribution parameters as  $\mu = 2.8ns, \sigma = 0.25ns$  for adders, and  $\mu = 7.5ns, \sigma = 1.5ns$  for multipliers. The characteristic delay of each function unit is then chosen as  $(\mu + \sigma)$  and the clock time is set to  $3ns$ . The initial scheduling and resource binding are done in a traditional HLS design flow using the ASAP strategy.

To demonstrate the timing yield improvement of our method, we compare the results after our proposed latch replacement against the original flip-flop based design. Table I shows the results of the timing

TABLE I

THE TIMING YIELD IMPROVEMENT AFTER LATCH REPLACEMENT

Name	Before	After	Abs. Impro.	Rel. Impro.
DES	64.8%	86.3%	21.5%	32.2%
FIR	82.1%	94.7%	12.6%	15.3%
AR	77.2%	92.8%	15.6%	20.2%
EWR	70.5%	91.8%	21.3%	30.2%
CHEM	68.3%	94.2%	25.9%	37.9%
Avg	72.6%	91.9%	19.4%	27.6%

yield comparison. The first column lists the benchmarks we used in the analysis. From the second column to the third column, we show the overall timing yield results before and after our latch replacement framework is applied, respectively. In the fourth and fifth columns, we compute the absolute improvement (Abs. Impro.) and the relative improvement (Rel. Impro.) over the original flip-flop design. As we can see from Table I, an average of 27.6% yield improvement could be achieved if we replace the storage element with latches in high-level synthesis.

To compute the area overhead after replacement, we first apply the resource re-sharing algorithm and count the increase of number of function units. We then add the area of all latches added to the design and deduct the area of all flip-flops removed from the design. For the sake of simplicity, we set the transistor count of each component as following [16]: a 32-bit CLA adder has 2598 transistors, a 16-by-16 bit multiplier has 19670 transistors, a 32-bit register implemented by flip-flop and by latch has 784 and 392 transistors, respectively. Table II estimates the area overhead introduced by our latch replacement. The second and third columns list the original resource usage, while the fourth and fifth columns show the usage after latch replacement. The seventh column lists the number of replaced flip-flops. The overhead on function units in the sixth column is compensated by the area saving on latches, resulting in the final overhead data in the last column. From the table we can see that, an average area overhead of 17.8% is introduced by the latch replacement. However, it has to be mentioned that, as the size of the design increases, the relative overhead reduces quickly from 34.5% of DES to 9.3% of CHEM. The decreasing trend indicates that our latch replacement approach will be applicable on large designs.

In Fig. 9, we compare our approach against the operation chained flip-flop design approach discussed in Section III and Fig. 2(b). The clock cycle time for the chained flip-flop design is set to 9ns (3 times of the cycle time in latch design). As operation chaining based design will also introduce resource overhead at the same time of improving timing yield, we define *Optimization Efficiency* as the yield improvement divided by the relative area overhead, and use it as a metric for comparing the efficiency of simultaneously maximizing timing yield and minimizing area overhead. As seen in Fig. 9, our latch based approach outperforms operation chained flip-flop design, especially in relatively large designs.

## VII. CONCLUSION AND FUTURE WORK

Increasing impact of process variations has led to a shift to statistical design methodologies across all levels of the design hierarchy. In this paper, we presented a new approach to mitigate the impact of process variations in high-level synthesis. We replaced flip-flops with latches to allow time slacks to be passed freely between control steps.

TABLE II

THE AREA OVERHEAD OF LATCH REPLACEMENT

Bench Name	Res. Before		Res. After		Res. Overhead	#FF Rep.	Total Overhead
	Add	Mul	Add	Mul			
DES	2	2	2	3	44.2%	11	34.5%
FIR	2	3	3	4	34.7%	25	19.4%
AR	3	3	4	4	33.3%	31	15.1%
EWR	3	3	4	4	33.3%	38	11.0%
CHEM	3	4	5	5	28.8%	43	9.3%
Avg	-	-	-	-	34.8%	-	17.8%

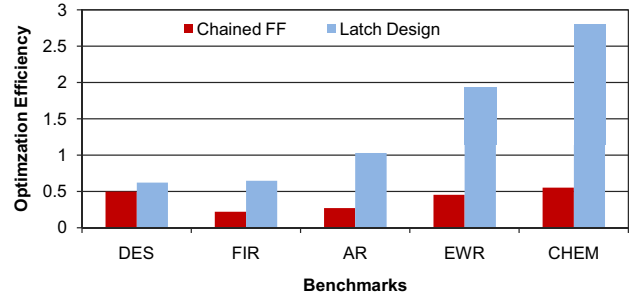


Fig. 9. Efficiency comparison of chained FF design and latch based design

Experimental results show that significant timing yield improvement can be achieved in our latch replacement framework. For future work, we are seeking for a thorough and analytical timing analysis of latch based design, in order to integrate the latch replacement into the optimization loop at the early stages of high-level synthesis.

## REFERENCES

- [1] A. Srivastava, D. Sylvester, and D. Blaauw. *Statistical analysis and optimization for VLSI: Timing and Power*. Springer, 2005.
- [2] S. Choi, B. C. Paul, and K. Roy. Novel sizing algorithm for yield improvement under process variation in nanometer technology. In *DAC*, pages 454–459, 2004.
- [3] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical clock skew analysis considering intra-die process variations. In *ICCAD*, 2003.
- [4] S. Raj, S. Vrudhula, and J. Wang. A methodology to improve timing yield in the presence of process variations. In *DAC*, 2004.
- [5] W. L. Hung, X. Wu, and Y. Xie. Guaranteeing performance yield in high-level synthesis. In *ICCAD*, pages 303–309, 2006.
- [6] J. Jung and T. Kim. Timing variation-aware high-level synthesis. In *ICCAD*, 2007.
- [7] David Chinnery, Kurt Keutzer, Jagesh Sanghavi, Earl Killian, and Kaushik Sheth. Automatic replacement of flip-flops by latches in asics. In *Closing the Gap Between ASIC and Custom*. Springer, 2007.
- [8] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Micro*, 2005.
- [9] A. P. Hurst and R. K. Brayton. of latch-based design under process variation. In *IWLS*, 2006.
- [10] Y. Lin and T. Wu. Storage optimization by replacing some flip-flops with latches. In *Euro-DAC*, 1996.
- [11] Yang Wooseung, Park In-Cheol, and Kyung Chong-Min. Low-power high-level synthesis using latches. In *ASP-DAC*, 2001.
- [12] P.G. Paulin, J.P. Knight, and et. al. Force-directed scheduling for the behavioral synthesis of asics. In *IEEE TCAD*, 1989.
- [13] Visweswariah Chandu and Ravindran Kaushik. First-order incremental block-based statistical timing analysis. *IEEE TCAD*, 99(99):1, 2005.
- [14] Raul Camposano and Wayne Wolf. *High-Level VLSI Synthesis*. Springer-Verlag New York, LLC, 2001.
- [15] Carlo Jacoboni and Paolo Lugli. *The Monte Carlo Method for Semiconductor Device Simulation*. Springer, 1990.
- [16] Carl Lemonds. A high throughput 16 by 16 bit multiplier for dsp cores. In *IEEE International Symposium on Circuits and Systems*, 1996.