Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Energy Engineering (36^{th}cycle)

Mix & Latch: High-Performance Designs with Single-Clock Mixed-Polarity Latches and Flip-Flops

By

Filippo Minnella

******

Supervisor(s):
Prof. Luciano Lavagno
Prof. Mihai Lazarescu

Doctoral Examination Committee:
Prof. Peter Beerel, *Referee*, University of Southern California, Los Angeles, USA
Prof. Sachin Sapatnekar, *Referee*, University of Minnesota, Minneapolis, USA
Prof. Jordi Cortadella, Universitat Politècnica de Catalunya, Barcelona, Spain
Prof. Alex Yakovlev, Newcastle University, Newcastle upon Tyne, UK
Prof. Mario Casu, Politecnico di Torino, Torino, Italy

Politecnico di Torino
2024

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Filippo Minnella
2024

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Abstract

Sequential circuits often use flip-flops (FFs) or latches for data storage. Latches have advantages in error-resilient applications, lower supply voltage operation, reduced power consumption, and increased operating frequency. However, complex timing constraints have limited their adoption in commercial and industrial designs. To overcome this limitation, researchers have explored the automatic conversion of FF-based designs into latch-based designs, primarily focusing on performance enhancement by reducing the clock period and considering potential area improvements.

Different solutions have been proposed, including pulsed latch designs, multi-phase clocking schemes and retiming methodologies. All of them have specific drawbacks that limit their deployment in industrial design flows which consist in: preventing pulse signal degradation in all operating conditions, increasing the area due to additional retimed registers, lack of fomal verification methodology or requiring multiple clocks generation and complex clock distribution networks.

In this thesis, we introduce a methodology called *Mix&Latch*, designed to address the mentioned limitations.

The key-points of the proposed flow are: trasforming flip-flop designs into positive transparent latches (PTLs) based designs that leverage time borrowing, incorporating negative transparent latches (NTLs) as retention barriers, instead of relying on delay padding, to address short-path hold constraints, employing a single clock tree throughout the design and merging adjacent latch pairs into positive-edge-triggered flops (PETFs) or negative-edge-triggered flops (NETFs) to reduce area overhead.

The first part of this thesis provides a comprehensive explanation of the proposed methodology. It describes the modeling of circuit timing and positional data, the conversion of the optimization problem into an integer linear programming (ILP) form and the enhancements made to the original methodology.

The second part of this thesis presents the experimental results obtained. In the initial version, experimental evaluations demonstrate the advantages of this approach on a suite of benchmark circuits. The enhancements to the algorithm are then evaluated on a *RISC-V* processor, showing a reduction in the implementation flow runtime, diminished area overhead, and enhanced timing performance in comparison to retiming, which was executed using a state-of-the-art commercial tool.

Contents

List of Figures

List of Tables

Chapter 1

Motivations

1.1 Challenges in sequential circuits design

Aggressive pipelining is common, in modern digital circuits, to push the maximum operating frequency. However, the addition of pipeline stages increases the number of registers and impact timing analysis complexity [1]. Furthermore, the addition of register stages caused by the extreme pipelining causes an increased use of resources translated in a higher area and power consumption.

Timing analysis ensures the correct operation of digital circuits at a specified frequency. These timing requirements can be categorized into two distinct classes of constraints: 1. Setup timing constraints: These constraints primarily relates to slow-propagating signals along the setup critical paths (SCPs). They are concerned with determining the maximum clock speed and rely on the assessment of the longest signal paths within the circuit. 2. Hold timing constraints: These constraints are centered around fast-propagating signals along the hold critical paths (HCPs). Their primary objective is to prevent early-sampling errors between pipeline stages and hinge on the evaluation of the shortest signal paths.

In the realm of design methodologies, those based on FFs are the most prevalent due to their advantages in terms of simplified timing analysis and robust support within commercial electronic design automation (EDA) tools. Regarding the aspect of hold timing constraints, the primary concern revolves around managing clock skew between consecutive stages. It is imperative for this skew to be less than the delay of the shortest signal path. In FF-based designs, this issue typically does not pose

a significant bottleneck. However, when employing aggressive pipelining, which entails increasing the number of sequential elements and consequently introducing a greater number of timing constraints, the complexity of the optimization problem can worsen. Setup timing constraints limit the maximum clock frequency of FF-based design. Splitting the logic in balanced pipeline stages, the longest paths are divided in shorter sub-paths, which timing is satisfied by smaller clock periods. However, exactly like for hold timing rules, the number of constraints to be satisfied increases with the addition of new pipeline stages.

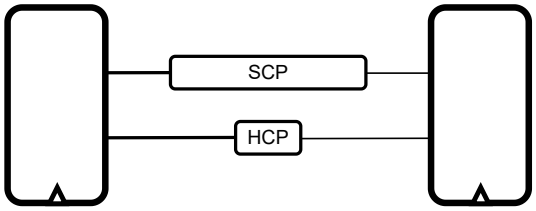


Fig. 1.1 FF-based design

Latches offer advantages, primarily because they can operate at higher frequencies, thanks to a feature known as time borrowing [2]. Time borrowing permits slow-propagating signals to traverse a pipeline stage even after the clock edge, a limitation that exists in FF-based approaches. However, it's important to note that latch-based pipelines with a single-phase clocking scheme suffer from stringent minimum timing constraints.

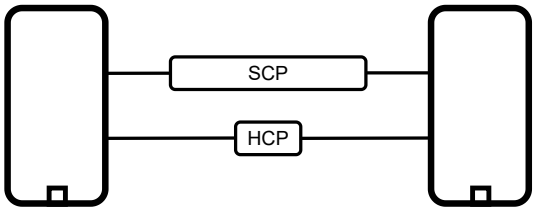


Fig. 1.2 latch-based design

The potential for an early-sampling error condition between consecutive registers is amplified due to the presence of time borrowing. This extended window offers fast signals a larger time window to generate the race condition [3]. As a consequence, single-phase latch-based designs are notably challenging to employ and are generally avoided in industrial applications. Nevertheless, the inherent challenges of increasing the number of pipeline stages in digital circuits, drive the exploration of innovative

design and implementation approaches aimed at enhancing performance without necessitating additional stages.

1.2 Latch-based designs

Sequential circuits use FFs or latches for data storage. Latches can be used in error-resilient applications [4, 5], work at lower supply voltages, reduce power consumption [6–8], and can increase operating frequency [9, 10]. Different approaches have been proposed to solve the problem of implementing digital circuits using latches instead of FFs. Because of the complexity in designing latch-based circuis at register transfer level (RTL) , it is more efficient to transform FF-based netlist in latch-based ones.

The seminal work of [11, 12] provides the formal definition of the system of equations which describe the timing behavior of a latch-based design. The authors show that the optimal timing analysis of a latch-based design can be performed by solving a system of equations similar to the one used for FF-based designs. After a linearization process, the system of equations is composed by linear equations and inequalities which can be solved using linear programming techniques. However, the increasing time to solve the linear programming model with bigger circuits strongly limit the applicability of this method.

A sub-group of solutions rely on generating a finite number of non-overlapping clocks, with optimized phases and duty cycles, fed to latches. The clocks definition is usually done using post-synthesis timing analysis. Zhang *et al.* [13] study the distribution of errors caused by sub-threshold voltage supply and propose a two-phase clocked latch-based method to solve the timing violations. Fojtik *et al.* [4] analyze a two clock-phase latch-based implementation of Razor flops to detect errors in an ARM Cortex-M3 processor. Cheng *et al.* [14] discuss a conversion algorithm using three clock phases to improve area and power consumption.

Another approach fully exploit duty cycle selection through the concept of *pulsed latch* introduced in [15–21]. These are composed by latches and pulse generators which shape the clock in input to latches in order to reduce their duty cycles to pre-computed value that respect min and max contraints. To limit the area cost, the pulse generators are shared by PTL groups and they are integrated either in a single

sequential cell for pulsed FF (P-FF) (P-Ls with the pulse generator within the latch cell) or in a cell containing multiple sequential blocks for pulsed registers (P-Rs).

Nevertheless, with shared pulse generators it is very difficult to prevent pulse signal degradation in all operating conditions [15], and the additional retimed registers for solving the remaining hold violations further increase the area.

Another noteworthy set of solutions is the retiming-based approach. Retiming is a technique that aims to achieve optimal balancing of combinational paths between two registers by relocating the sequential element along the combinational paths. This concept bears resemblance to pipelining but is executed automatically during the implementation process, eliminating the need for manual intervention by the designer. In most cases, flip-flops (FF) are modeled using the master-slave approach, which employs two latches operating at opposite clock level polarities. Retiming-based approach moves master and slave latches in order to balance the combinational paths between them. These pairs of latches are working at opposite phases [22] avoiding the addition of complexity to the min-timing constraints because, exactly as in FF-based solutions, the requirement depends on the clock skew and the shortest path delay without the time borrowing addition. The additional register stages introduced by retiming can increase the complexity of the timing analysis and the area of the design. Yoshikawa *et al.* [23] present a single-phase forward retiming algorithm for FF-based design conversion, using commercial tools for retiming. Hassan *et al.* [7] and Singh *et al.* [6] present implementation flows to transform FF-based designs into latch-based or mixed designs. In almost all previous cases, the optimization uses post-synthesis timing information that may substantially differ from the post-layout one, thus potentially leading to grossly sub-optimal post-layout performance. Furthermore, [7, 23] evaluate the performance only on post-synthesis data, thus ignoring the place and route (P&R) overheads.

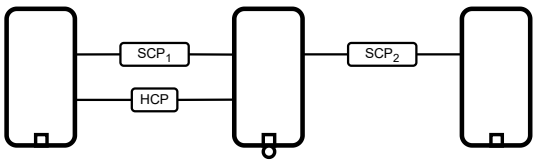


Fig. 1.3 *dual_latch*-based design

1.3 Problem statement

Latch-based circuits have the potential to enhance the performance of digital circuits by leveraging time borrowing, which allows longer paths to navigate through a pipeline stage. Despite these advantages, their adoption in commercial solutions is limited due to the intricate timing requirements stemming from stringent hold timing constraints imposed by the same time borrowing mechanism that boosts frequency. Various approaches have been proposed to address these challenges, often beginning with the formulation of the clocking problem using clocks featuring different duty cycles and phases. However, these approaches exhibit drawbacks that hinder their applicability within industrial implementation workflows. Multi-phase solutions necessitate the generation of multiple clocks, introducing a substantial overhead. On the other hand, pulsed-latch based solutions entail the inclusion of on-chip pulse generators, which result in added area overhead and complexity in ensuring the signal integrity of the numerous clocks supplied to the sequential elements. In contrast, retiming-based solutions have achieved state-of-the-art performance without complicating the implementation process. Nevertheless, since they involve relocating sequential elements within the circuit, they do not provide guarantees for formal verification and design correctness.

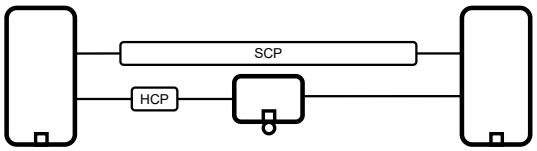


Fig. 1.4 *Mix & Latch*-based design

This thesis introduces a methodology known as *Mix & Latch*, which effectively tackles the issues discussed. This approach converts at first the FFs-based netlist in a single-phase latch-based circuit, and within it, places retention barriers to address the intricate minimum-timing constraints, thus alleviating the complexities in the implementation process. These retention barriers comprise latches that operate at the opposite polarity compared to the functional ones. Notably, there's no necessity to generate additional clocks because the retention barriers are synchronized with the same clock used by the functional latches. Importantly, the proposed methodology ensures the accuracy of the design and the feasibility of formal verification, as it maintains the original register positions intact.

Chapter 2

Preliminary analysis

2.1 Timing analysis

The timing analysis process in sequential circuits defines the critical paths and helps ensuring the correct sampling of the signals generated by the combinational logic. Depending the nature of the sequential elements, the analysis is different. For each different nature, the hold and setup constraints must be defined.

The simplest case is the one of the *flip-flops*, which are edge-triggered elements, i.e. the sampling of the data is performed at the rising or falling edge of the clock signal. Given a flip-flop (FF) j the longest path in input to the cell ($\overline{d_{i,j}}$) contributes to the clock period (T) lower bound:

$$T - T_{setup_j} \geq \max_i \left\{ \overline{d_{i,j}} - \Delta_i \right\} \tag{2.1}$$

with Δ_i equal to the maximum clock-to-Q delay of the sequential element i. The shortest path in input to the cell ($\underline{d_{i,j}}$) do not introduce bounds to the clock in FFs-based systems, however it is important to consider it in order to avoid the early sampling of the signal:

$$T_{hold_j} \leq \min_i \left\{ \underline{d_{i,j}} + \delta_i \right\} \tag{2.2}$$

with δ_i equal to the minimum clock-to-Q delay of the sequential element i.

In latch-based circuits, that are level-sensitive elements, the analysis is more difficult. As already discussed in the previous chapter they allow for higher performance

even though the possible early sampling caused by the transparent phase can lead to an error in the dataflow. The seminal works [11, 12] formalize the problem of the level-sensitive sequential circuits clocking, taking also into account the possible phase shifts between the sampling stages, and provide a *linear programming* model to find a solution to the optimal period clocking problem. The first additional degree of complexity is related to the computation of the maximum (D_i) and minimum (d_i) departure time of the signal from the launching latch i. They can happen in any time instant of its transparent phase, defined by the latch phase x_i:

$$D_i = \max_i \{A_i, x_i\} \tag{2.3}$$

$$d_i = \max_i \{a_i, x_i\} \tag{2.4}$$

Starting from these definitions, the minimum (a_j) and maximum (A_j) arrival times of the signal at the capturing latch:

$$A_j = \max_{i,j} \left\{ D_i + \Delta_i + \overline{d_{i,j}} + x_i \right\} \tag{2.5}$$

$$a_j = \min_{i,j} \left\{ d_i + \delta_i + \underline{d_{i,j}} + x_i \right\} \tag{2.6}$$

The setup Eq. (2.7) and hold Eq. (2.8) constraints now takes into account the transparent phase of the latches. Considering the *time borrowing* on the positive clock level, the setup constraints become less stringent by a factor of duty cycle (DC).

$$T \cdot (1 + DC) + x_j \geq A_j + T_{setup_j} \tag{2.7}$$

And the hold constraints worsen by the same factor:

$$T \cdot (DC) + x_j \leq a_j - T_{hold_j} \tag{2.8}$$

2.2 Clock skew

In sequential circuits, each sequential element is connected to a clock signal which is used to synchronize the operation of the circuit. During the synthesis step, in circuits using edge-triggered flip-flops, the optimization focuses in optimizing the logic paths in order to reduce the delay of the longest paths. However, the physical synthesis of the clock tree introduces delays in the arrival time of the clock signal at the different sequential elements. Considering two flip-flops *i,j* connected by combinational logic, and their phases x_i, x_j, the clock skew is defined as the difference of the two phases:

$$\Delta x_{i,j} = x_j - x_i \tag{2.9}$$

These unbalances in the sampling instant of the sequential elements introduce the possibility of early sampling, *hold constraints violation*, or late sampling, *setup constraints violation*, of the data.

The first approaches to the implementation of the clock tree tried to minimize the clock skew, *zero skew* [24], in order to meet the hold and setup analysis performed at synthesis time. However, by correctly scheduling the clock phases [25–27], *fixed skew*, it is possible to achieve a lower clock period thanks to cycle borrowing, i.e. delaying the capturing time for long paths. Fig. 2.1 shows an example of useful skew applied to a sequence of pipeline stages.

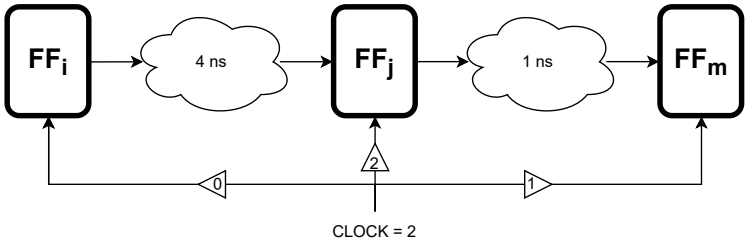


Fig. 2.1 Useful skew example

[27] proposes a method to find the minimum clock period of a circuit computing if there is a possible clock schedule that satisfies the timing constraints. Given a circuit composed of N sequential elements, the clock period T, the clock phases x_i, x_j of two sequential elements *i,j*, two inequalities are defined to avoid the early and late sampling of the data. To avoid late sampling the capturing time of the sequential element *j* should be greater than the latest arrival time ($\overline{d_{i,j}}$) of the data:

$$x_j + T \geq x_i + \overline{d_{i,j}} + T_{setup} \tag{2.10}$$

To avoid early sampling instead, the capturing time of the sequential element j should be less than the earliest arrival time ($\underline{d_{i,j}}$) of the input data:

$$x_j \leq x_i + \underline{d_{i,j}} - T_{hold} \tag{2.11}$$

Considering the two inequalities, it possible to formulate the minimization of the clock period problem as:

$$\begin{aligned}
\text{minimize} \quad & T && (2.12)\\
\text{subject to} \quad & x_j - x_i \geq \overline{d_{i,j}} - T + T_{setup} && (2.13)\\
& x_i - x_j \leq T_{hold} - \underline{d_{i,j}} && (2.14)\\
& \forall i \in \{1,\ldots,N\} | x_i | \leq T && (2.15)
\end{aligned}$$

The constraints 2.10 and 2.11 constitutes the first two inequalities of the problem, the last one ensures that the maximum phase shift is lower than the clock period.

To solve this problem, [27] proposes a binary search algorithm which iterates on the clock period, tracing the minimum and maximum limit of the search interval and looking if the minimum proposed value for the clock period is feasible or not. If a specific clock period is feasible then the maximum value of the considered interval will be updated, if not then the minimum value will be updated. The feasibility check is done through a Bellman-Ford like algorithm that looks for *positive cycles* after the clock schedule.

To reduce the number of iterations, the low and high bounds are defined. The minimum bound (T_{low}) is computed as:

$$T_{i,j} = \overline{d_{i,j}} - \underline{d_{i,j}} + T_{setup} + T_{hold} \tag{2.16}$$

$$T_{low} = \max_{i,j} \left\{ T_{i,j} \right\} \tag{2.17}$$

and depends on the time window in which the data can be sampled.

The maximum bound (T_{high}) is defined as:

$$T_{high} = \max_{i,j} \left\{ \overline{d_{i,j}} + T_{setup} \right\} \tag{2.18}$$

and depends on the maximum delay of the combinational paths. The iterative search algorithm is shown in Algorithm 1.

Algorithm 1 Binary search algorithm for the clock period

```
1:  T_min ← T_low
2:  T_max ← T_high
3:  while T_max − T_min > ε do
4:      T ← (T_min + T_max)/2
5:      feasible ← BELLMAN-FORD(T)
6:      if feasible then
7:          T_max ← T
8:      else
9:          T_min ← T
10:     end if
11: end while
```

Given the computed T_{max}, any clock period $T \geq T_{max}$ is feasible. To evaluate the *Bellman-Ford* algorithm, the circuit is represented as a graph where the vertices are the clock phases of the sequential elements and the edges are the relations defined by 2.10 and 2.11. Figure 2.2 shows the constraints graph of an example circuit.

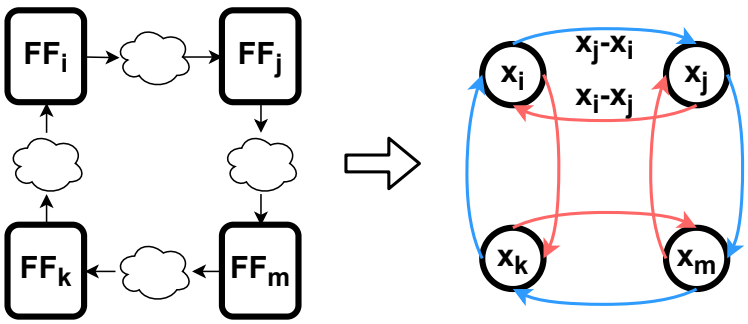


Fig. 2.2 Constraint graph of the circuit

Another problem which should be considered when optimizing the skew scheduling is the *wave pipelining*, which is the coexistence of two different propagating values in the same combinational path caused by the unbalanced sampling time at different stages of the pipeline. As proposed by [28], starting from the model

proposed in [25] specific constraints should be applied to avoid the overlap of the two propagations.

2.3 Retiming

Retiming is a technique that aims to achieve optimal balancing of combinational paths between two registers by relocating the sequential element moving them across the logic gates. This technique do not change the number of registers on a path

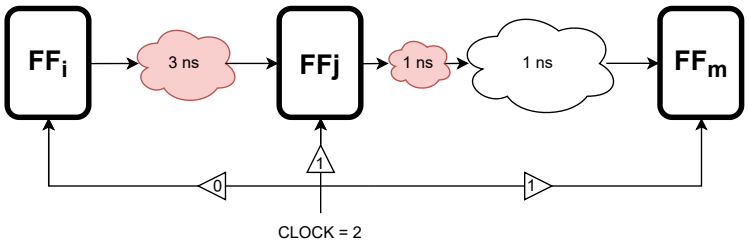


Fig. 2.3 Retiming example

from primary inputs to primary outputs, thus not changing the total latency of the considered block or the behaviour of the circuit.

It could lead to further optimization in the delay of the combinational logic because it changes the boundaries of the combinational paths. Retiming could be applied to optimize multiple metrics, this work introduces only the clock period minimization. It can be applied to both edge-triggered based sequential circuits and level-sensitive based ones.

The *ASTRA* approach [29] shows the relations between retiming and clock skew scheduling in minimum period optimization. The example figures 2.1 and 2.3, describes the same circuit with different optimization techniques applied; the first one shows how increasing the latency applied to the FF_j to *2* reduces the clock period to *2* units, the second one shows how moving the FF_j backward in the combinational logic reaches the same result with a lower delay x_j. From the examples, the equivalence between the two methods is clear. The reduction in the clock latency is equivalent to the time delay of the combinational logic moved to the second stage of the pipeline.

The formal definition from [29] describes the equivalnce as:

- *Backward Retiming*: moving a register backward in the combinational logic corresponds to decreasing the delay of the clock signal feeding it.
- *Forward Retiming*: moving a register forward in the combinational logic corresponds to increasing the delay of the clock signal feeding it.

The generalization of the equivalence to a multi-input and multi-output combinational circuit is straightforward:

- Given a circuit that has a double sampling stage at the input, one of the two stages is forward retimed to the output. For each sequential element j retimed to the output, the delay of the clock signal feeding it is:

$$x_j = \max_i \left\{ x_i + \overline{d_{i,j}} \right\} \tag{2.19}$$

  Which is equivalent to the clock phase able to satisfy the setup constraints at the output of the combinational block.

- Similarly, given a circuit that has a double sampling stage at the output and one of the two stages is backward retimed to the input. For each sequential element k retimed to the input, the delay of the clock signal feeding it is:

$$x_k = \min_j \left\{ x_j - \overline{d_{k,j}} \right\} \tag{2.20}$$

  Which is equivalent to the clock phase able to satisfy the setup constraints.

The *ASTRA* approach uses this equivalence to efficiently perform the retiming of the circuit. The algorithm looks for a feasible clock schedule for the sequential elements of the circuit, and then moves the registers accordingly over the logic. Once that a register is moved, the clock schedule is updated accordingly. If at the end of the procedure the clock delays of all the flip-flops are zero, then the optimal clock period is reached.

2.4 Analyzing different optimization models

As discussed in the previous sections, it becomes evident that optimization through skew scheduling yields the lower bound for period minimization, which extends

to the retiming problem as well. Utilizing the methodology proposed by [27] for determining the lower bound of the clock period also results in optimal performance for the retiming problem. The presented *constraint graph* considers both setup and hold constraints to enable the determination of the scheduling of various clock phases. It's essential to note that the combinational delay of the most critical cycle has a finite margin for optimization, thus establishing a limit on the circuit's maximum frequency. Fig. 2.4 illustrates the critical paths within the same constraint graph, excluding the edges accounting for the short paths.

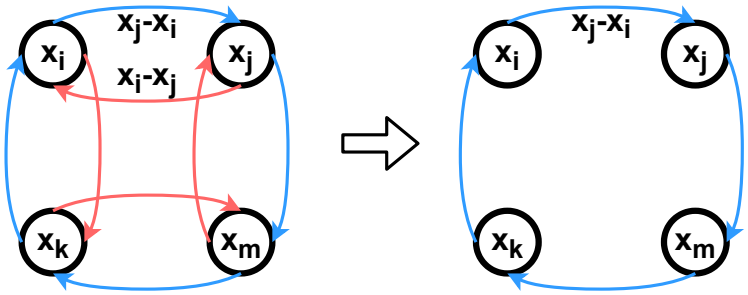


Fig. 2.4 Critical paths of the circuit

This configuration serves the purpose of determining the lower bound for the clock period using the same database that was initially computed for the original algorithm. If a disparity arises between the maximum frequency identified in the initial configuration and the frequency determined by exclusively considering setup critical paths, the variance is attributed to the hold critical paths. A remaining scope for improvement exists that cannot be fully realized through clock skew adjustments and retiming alone. A potential approach to address this issue involves introducing delay padding to the combinational logic, specifically on the segments of the hold critical paths that do not overlap with setup critical paths. The work by [30] introduces a linear programming (LP) model to identify the optimal amount of delay padding required on short paths to mitigate the issues related to hold constraints.

To evaluate the potential benefits of this approach for optimizing the clock period, one possible solution is to incorporate the delay padding into the constraint graph and employ the Bellman-Ford algorithm to determine the circuit's maximum frequency. Fig. 2.5 depicts the constraint graph with delay-padded paths, extending up to a predetermined value equal to the high time of the clock cycle ($DC \cdot T$).

This model is valid for both FF-based and latch-based circuits. Despite the information presented in Eq. (2.7), the setup constraints for latch-based circuits are

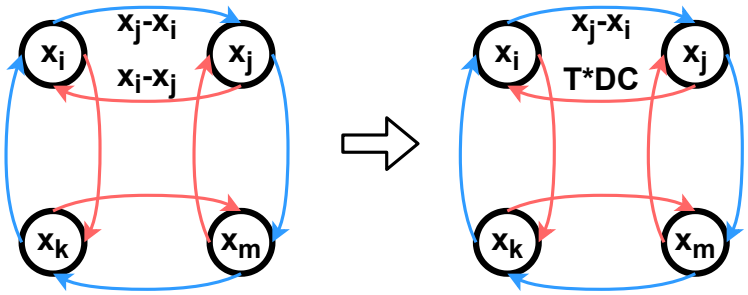


Fig. 2.5 Delay padded circuit

the same of FF-based approach. The maximum setup optimization through latch time borrowing overlaps with the one obtainable by retiming. As shown by Eq. (2.8) latch-based circuits, the main problem is the early sampling of the data caused by the extended transparent phase of the clock signal. The delay padding enables solving this problem and allows the scheduling of the clock. The table 2.1 shows the maximum frequency of the different configurations for the circuits considered in this work. The duty cycle for this analysis is set to *50%*.

Circuit	**Original** (ns)	**Setup Opt.** (ns)	**Delay padded** (ns)
s1196	0.182	0.182	0.218
s1423	0.469	0.469	0.469
s5378	0.324	0.324	0.324
s15850	0.739	0.645	0.645
s38584	0.632	0.632	0.632
des3	0.729	0.486	0.646

Table 2.1 Maximum frequency of the configurations

The results indicate that with the exception of two cases, namely *s15850* and *des3*, the primary limiting factor for the maximum circuit frequency are the setup critical paths. This observation relies from the fact that both the analysis of the original constraint graph and the configuration focusing solely on setup critical paths yield identical outcomes. The delay padding approach is capable of achieving, at a minimum, the same frequency as the original configuration, except in the case of *s1196*. In the context of the *s15850* and *des3* circuits, the delay padding approach surpasses the original configuration's frequency. This superiority is attributed to the constraints imposed by the hold critical paths. Consequently, this benchmarking exercise highlights the need to explore solutions that can assist the scheduling of clock delays in order to attain higher frequencies.

Chapter 3

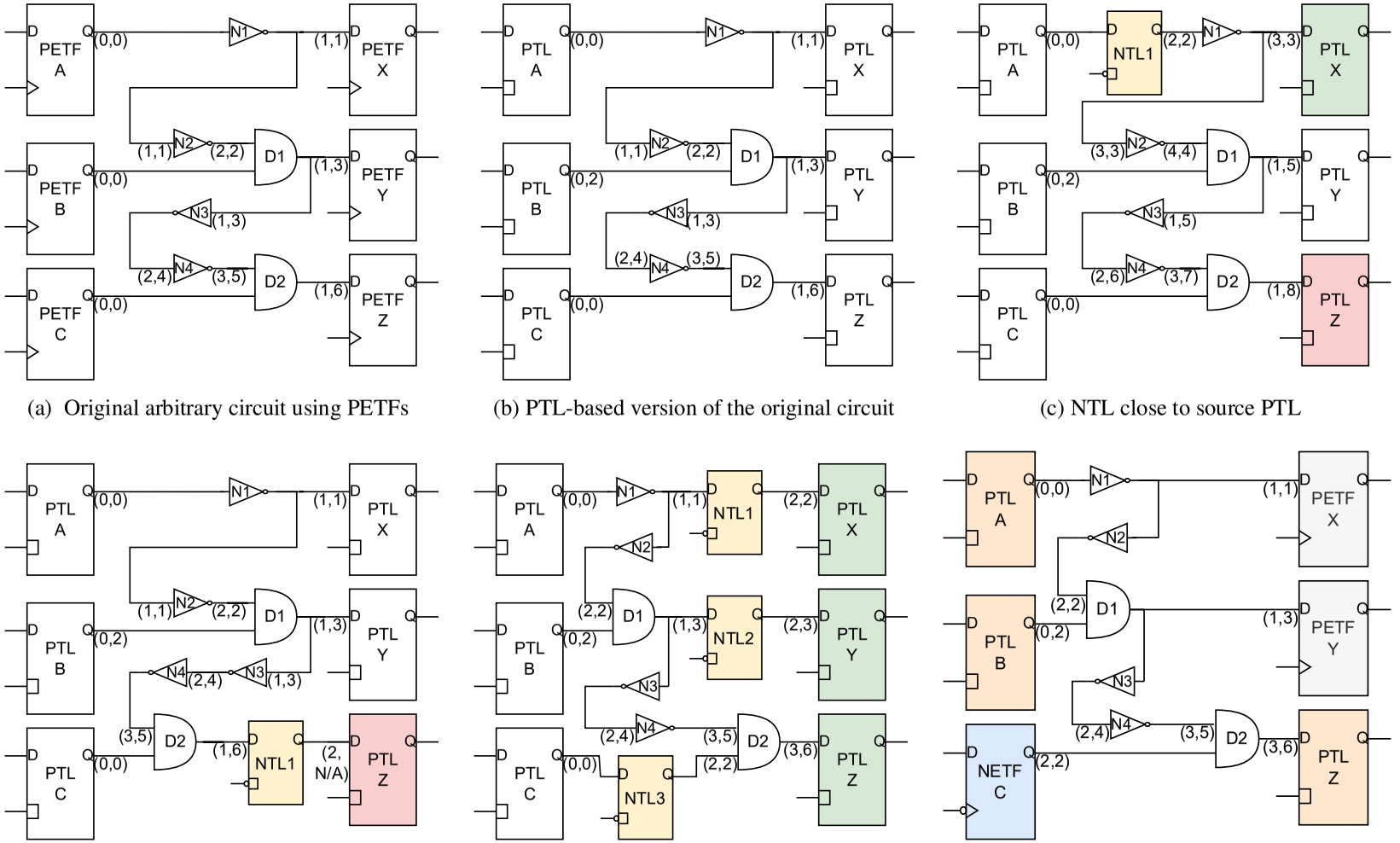
Methodology

3.1 Original flow

This chapter proposes the *Mix&Latch* method, which uses a conventional 50 % duty cycle (DC) single-phase clock. Hold time violations are solved by inserting NTLs driven by the same clock tree as the PTLs. First *the resulting clock period is optimized* by combining time borrowing and NTL retiming. Then, as a *secondary objective, area recovery is used to reduce the area overhead* by creating NTL-PTL sequences whenever possible. These primary/secondary pairs are then converted into either PETFs or NETFs, thus obtaining an optimized *mixed* design with PTL, PETF, NTL, and NETF sequential elements. *Mix&Latch* also preserves a sequential element in each of the original FF locations. This enables a 1-to-1 mapping from FFs to sequential elements and ensures that equivalence checking can be performed using conventional methods comparing combinational clouds.

Fig. 3.1 shows the application of this optimization algorithm to a simple case [1]. The original arbitrary PETF circuit is shown in Fig. 3.1a. All pins are annotated with a parenthesized number pair indicating the (*minimum arrival time at pin p* ($AT_p^{\min}$), *maximum arrival time at pin p* ($AT_p^{\max}$)). For simplicity, the assumption in this figure is of unitary delays for combinational gates, zero delays for the sequential elements, and zero setup/hold FF constraints, while this algorithm uses delays from timing analysis. In a PETF-based circuit, the minimum clock period, $T_{\min}$, is set to the

[1] The circuit in Fig. 3.1 do not account for interactions with I/O pins because they are always modeled as being sampled by FFs and this would increase the complexity of the example.



(a) Original arbitrary circuit using PETFs

(b) PTL-based version of the original circuit

(c) NTL close to source PTL

(d) NTL close to sink PTL

(e) NTL-based solution

(f) Solution with merged resources: PTL-to-NTL yields NETF, NTL-to-PTL yields PETF

Fig. 3.1 Optimization algorithm applied to an arbitrary positive-edge-triggered flop (PETF) circuit (3.1a). Conversion to a circuit based on positive transparent latch (PTL) (3.1b). Optimization for sequences of PTL-to-negative transparent latch (NTL) (3.1c) or NTL-to-PTL (3.1d). Conversion of complementary latch sequences (3.1e) into positive-edge-triggered flops (PETFs) (3.1f) or NETFs. Pins are annotated with (*minimum arrival time* ($AT^{\min}$), *maximum arrival time* ($AT^{\max}$)). Gate delays are unitary and sequential delays are zero.

longest maximum arrival time at the endpoint pin ($AT^{\max}_{\text{end}}$), hence $T_{\min} = 6$ in this example. After the PTL conversion shown in Fig. 3.1b, the circuit can use time borrowing up to half a clock period (for $DC = 50\,\%$) for $AT^{\max}_{\text{end}}$

$$AT^{\max}_{\text{end}} = T_{\min}\,(1 + DC) \Longrightarrow T_{\min} = \frac{6}{1 + 0.5} = 4. \tag{3.1}$$

Note that there is an additional critical path with delay 6, PTL B $\rightarrow$ PTL Z, due to time borrowing at PTL B.

Despite the desirable $T_{\min}$ reduction by 33 % compared to the PETF version, there are hold violations at the inputs of PTLs X, Y, and Z because their minimum arrival time ($AT^{\min}$) is lower than the positive pulse width, $PPW = DC \cdot T_{\min}$. To solve the hold violations, a group of nets is selected using the mathematical model described below, and an NTL is placed in front of the endpoint pin of the selected nets. As the NTLs become transparent after the positive pulse, they guarantee a delay longer than the PPW for all paths.

However, the added NTLs can reduce performance. For example, Fig. 3.1c shows that while placing the NTL too close to the source PTL solves the hold violation at the input of PTL X, the additional delay causes a setup violation at the input of PTL Z, which now requires a longer period, $T > T_{\min}$. Fig. 3.1d shows that a NTL can solve the hold violations at the input of PTL Z, but it causes a new setup violation at the input of the NTL that closes at T. Hence, the signal cannot reach PTL Z in time, which also requires a longer $T > T_{\min}$.

This algorithm optimizes the position of NTLs to reach a solution that, as shown in Fig. 3.1e, solves all hold violations without performance penalty, under the assumptions discussed in Section 3.1.4.

Next, adjacent NTL-PTL pairs are merged as PETF to reduce the area, and PTL-NTL pairs as NETF, as shown in Fig. 3.1f. This solution has the same $T_{\min}$ and area as the one in Fig. 3.1b, no hold violations, and uses the same number of sequential elements as the original version. In some cases part of the latches cannot be merged, which leads to area penalty (discussed in the experimental results). In other cases the PTLs do not need hold time fixing, yielding both faster and smaller circuits.

Several works propose design optimization using a mix of PETF and PTL. Here there are described the main ones, in order to set the stage for this chapter. Hassan

et al. [7] propose to start from an FF-based netlist, analyze sequences of three FFs, and replace the middle one with a PTL retimed to match the timing constraints. This approach seems to increase the clock frequency, reduce the power consumption and the cell area, but the experimental data cover only logic synthesis, without considering placement and routing. Furthermore, equivalence checking may be more difficult because retiming changes the original position of the sequential elements [31].

Singh *et al.* [6] describe a retiming method to generate a PTL-NTL-based netlist starting from a FF-based one. Because the synthesis tools have poor support for latch retiming, they propose replacing the primary/secondary latches with FF pairs, doubling the frequency and finally retiming the design using a commercial tool. Although they focus on reducing the power consumption, the results are poor in terms of both power and area because the algorithm is effective for only one frequency due to a sub-optimal retiming strategy. Moreover, experimental results are shown only for one circuit.

The main contributions to the state-of-the-art are:

- A two-step implementation flow to obtain a working layout for an optimized version (Fig. 3.1f) of the PETF-based netlist (Fig. 3.1a). The implementation is fully based on commercial EDA tools and fully exploiting useful skew, both in the baseline against which this chapter compares and in the obtained results.
- A methodology to reduce the sequential resources and generate the NTL allocation, using post-layout timing data and exploiting incremental placement and routing starting from the post-layout netlist. The NTLs work as retention barriers for signals in short paths, reducing the hold constraints complexity. To recover area, the PTL-NTL pairs are merged into FFs.
- Maintaining a 1-to-1 correspondence between each original FF and a FF or a latch in the final circuit, to allow equivalence checking for design verification with traditional tools.

3.1.1 *Mix & Latch* Optimization Flow

Fig. 3.2 shows the optimization flow, which starts from a RTL description and produces a layout with mixed sequential resources. It includes four main steps:

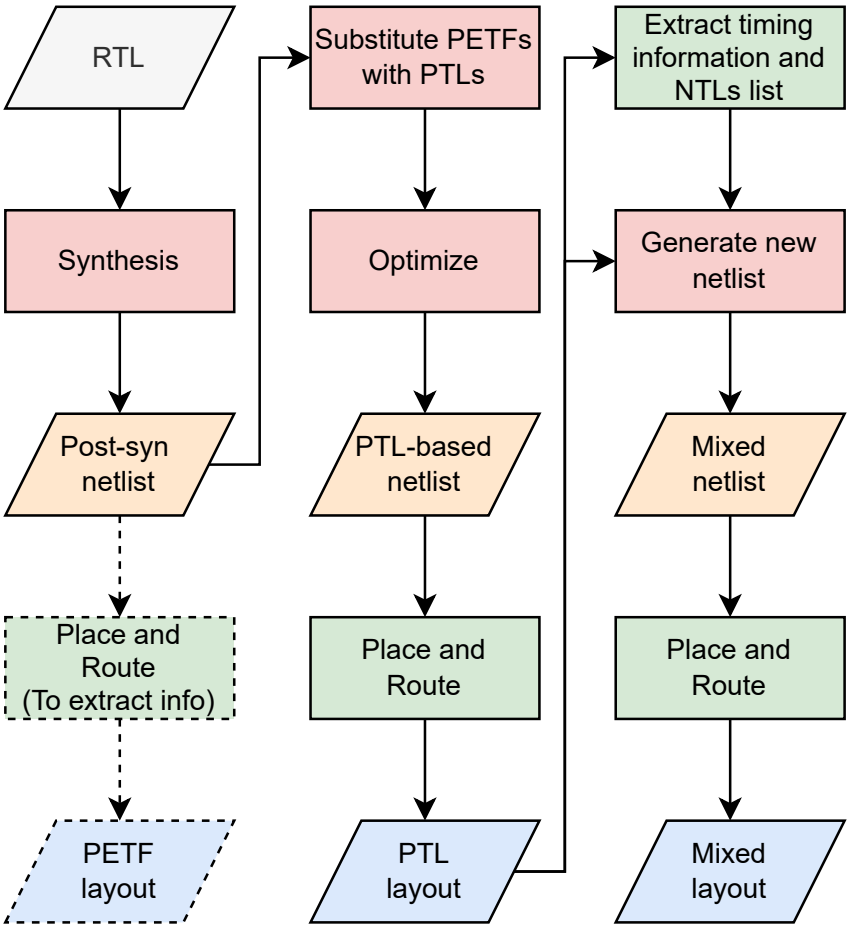


Fig. 3.2 Implementation flow starting from the register transfer level (RTL) description using positive-edge-triggered flops (PETFs), positive transparent latches (PTLs), and negative transparent latches (NTLs). Synthesis steps are in red, post-synthesis netlists in orange, layout steps in green, and post-layout netlists in blue. The PETF layout is only used to provide the baseline results.

- Generate the PTL-based layout by replacing all sequential elements with PTLs (shown in the second column in Fig. 3.2 and discussed in Section 3.1.2).
- Create a graph representation of the timing and positional information extracted from the PTL-based layout (discussed in Section 3.1.3, Section 3.1.4, and Section 3.1.5).
- Define the circuit location of NTLs, PETFs, and NETFs using an ILP formulation, and inserting them in the PTL-based netlist.
- Generate the layout of this new circuit (see the right column in Fig. 3.2 and the discussion in Section 3.1.6).

The NTL selection using ILP is similar to backward retiming [23] in a primary/secondary FF netlist. However, the formulation and graph representation are different because they consider the post-layout timing data and avoid the redundant NTLs. The designs are synthesized and implemented at several clock frequencies to determine iteratively the highest possible operating frequency for both the mixed design and of the PETF design.

We leave to future work the in-depth analysis of design for testability (DFT) needed for the practical adoption of our methodology. We note however that DFT can be implemented with traditional tools by adding some scan-only NTLs to the PTLs [32].

Retiming techniques have the drawback that equivalence checking for design verification cannot be solved in a reasonable amount of time even for relatively small circuits, such as the `s38584` from the ISCAS benchmark [31], which we also use in our experiments as shown in Section 4.1. *Mix & Latch* does not have this problem because it preserves a 1-to-1 correspondence with the FFs in the original design using either FFs or PTLs. The 1-to-1 correspondence also helps solving the initialization problem for the netlist, i.e., finding a consistent initial value of the circuit registers that maintains the circuit equivalence [33].

3.1.2 Positive transparent latch-based circuit

The first processing step generates the PTL-based layout. The RTL description of the target design is synthesized using a commercial tool. The considered circuits

have only PETFs to ease the analysis, but the same methodology can be extended to circuits based on NETFs or mixed. Once the netlist is synthesized, all the PETFs are replaced with PTLs using the same commercial tool. Because cell resizing will be automatically done by the layout tool (if needed), the PETFs are replaced with the smallest PTLs from the technological libraries.

The netlist modified this way is provided to the layout tool, which produces the post-P&R design. Unlike [11, 12], all hold constraints are temporarily ignored (using a standard design constraint command of the tool) to obtain a layout of the PTL-based netlist that meets the setup constraints.

The generated layout thus potentially violates hold conditions, which will be solved afterwards.

3.1.3 Graph model

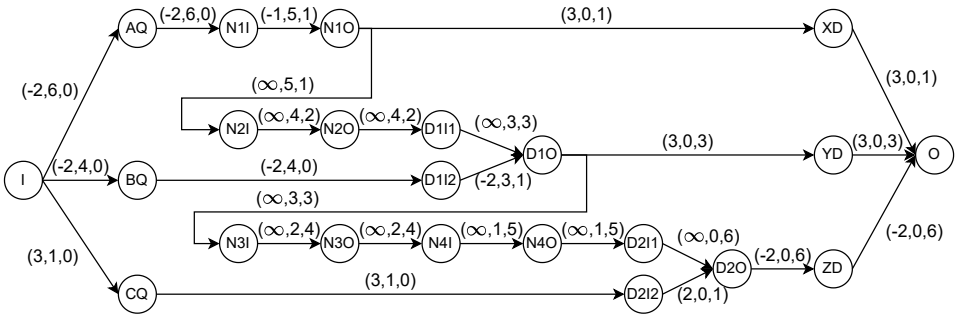
The state-of-the-art circuit graph representations [34, 12] are not suitable for our optimization algorithm because they either exclude the sequential elements [34], or aggregate pin data for the worst case delay [12].

For our method, the circuit is represented as a graph (V,E), where V represents the set of all pins and I/O ports and E the connections (wires or cells) between them. The nets and pins of the clock tree are not included. All sampling of input and output ports is defined as synchronous, coinciding with the rising edge of the system clock. This means that no borrowing of time from the environment is permitted and the new circuit uses the same boundary conditions as the original one.

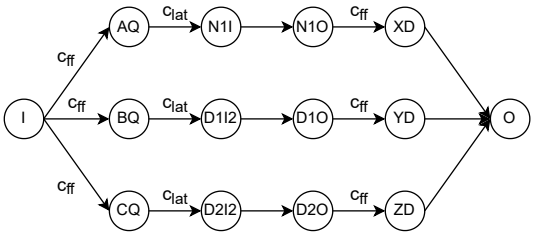
Fig. 3.3 shows an example of two graphs which are discussed later. Static timing analysis (STA) timing information is a three-value tuple associated to graph edges (see Section 3.1.4), while latch location is a value associated to edges of a different graph (see Section 3.1.5).

3.1.4 Timing Graph

We create a timing graph (TG) that drives our optimization algorithm to limit the setup slack (SS) degradation due to potential NTL insertions before gate input pins in the PTL-based netlist. The computation of the edge attributes of this graph



(a) : Timing graph (TG)



(b) : Short-path subgraph (SPG)

Fig. 3.3 Graph generation: (a) TG of the example in Fig. 3.1b. The attribute is a 3-tuple with elements computed using Alg. 2, Eqn. (3.6), and Eqn. (3.5), respectively. The first value is an estimation of the setup slack caused by NTL insertion, the second and third values are the lengths of the sub-paths generated by NTL insertion. (b) SPG of the same circuit: it has fewer edges and vertices than TG because it considers only pins and connections that belong to short paths.

uses the timing data extracted from static timing analysis (STA), shown in Fig. 3.4, Fig. 3.5, and Table 3.1. Unlike Fig. 3.4, all arrival times are obtained from the STA considering the clock latencies of the PTL-based layout. The pin (p) for which STA extracts the info is the endpoint of the edge (e) to which the related attribute is associated. From the STA timing data we obtain three values: (1) the *Estimated Setup Slack for pin p* (ESS_p), (2) the *p to SCP$_p$ endpoint delay* (D^p_{ptl}), and (3) the *SCP$_p$ startpoint to p delay* (D_p^{ptl}). These three values are assigned as a 3-tuple attribute to the edges of timing graph (TG).

Estimated Setup Slack

The Estimated Setup Slack (ESS_p) is computed for each edge (e) endpoint pin (p) using Algorithm 2, which estimates the value of the setup slack (SS) related to the pin (p) if an NTL were placed in front of it. It receives in input the timing info from STA for the considered pin and returns the attribute ESS_p. It is important to highlight

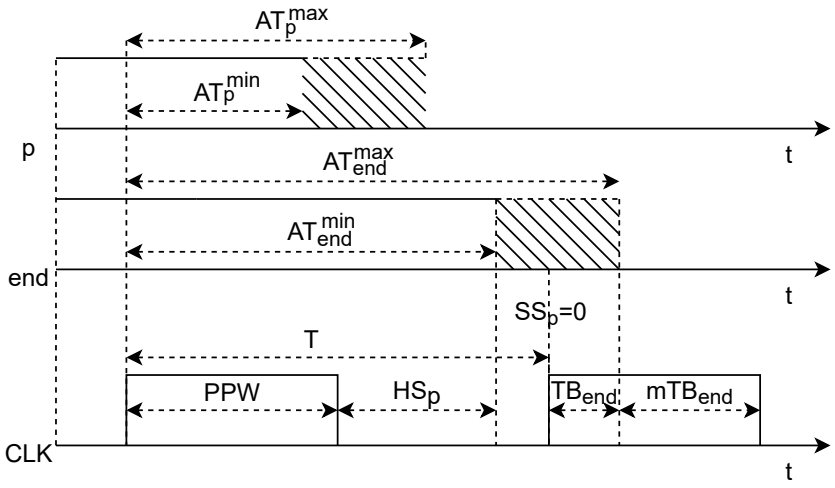


Fig. 3.4 Timing diagram, showing the arrival times and the slacks related to an ideal clock. During attribute computations the clock latencies are real and referred to the clock tree built in the PTL-based netlist. Because $AT^{\max}_{\text{end}}$ arrives during the PTL transparent window, SS_p is 0.

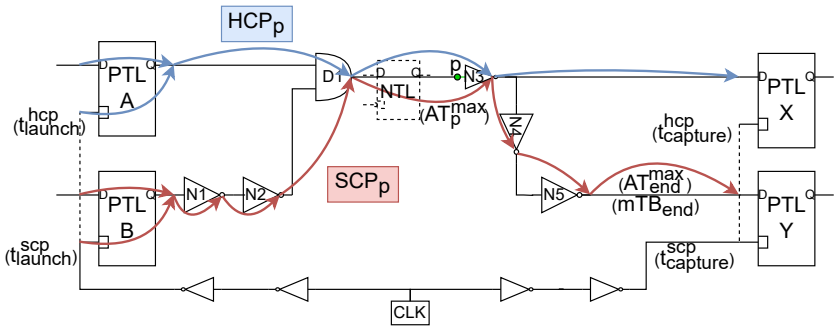


Fig. 3.5 Circuit showing how the information from the STA tool is extracted.

Table 3.1 Timing computed by Algorithm 2 with data from static timing analysis

NAME	DEFINITION
HCP_p	Hold critical path passing through pin p
SCP_p	Setup critical path passing through pin p
$t^{\mathrm{HCP}}_{\mathrm{launch}}$	Clock latency of the considered HCP_p launching PTL
$t^{\mathrm{SCP}}_{\mathrm{launch}}$	Clock latency of the considered SCP_p launching PTL
$AT^{\max}_p$	Maximum arrival time at pin p
$AT^{\min}_p$	Minimum arrival time at pin p
SS_p	Setup slack at pin p
HS_p	Hold slack at pin p
$t^{\mathrm{HCP}}_{\mathrm{capture}}$	Clock latency of the considered HCP_p endpoint PTL
$AT^{\max}_{\mathrm{end}}$	Maximum arrival time at endpoint PTL
$AT^{\min}_p$	Minimum arrival time at endpoint PTL
mTB_{end}	Time borrowing margin at endpoint PTL of SCP_p
TB_{end}	Time borrowing at endpoint PTL of SCP_p
PPW	Clock positive pulse width

that the computation of $AT^{\max}_p$ takes into account the possible time borrowed by the launching PTL. It relies on the following assumptions:

- To estimate the SS_p degradation caused by the NTL insertion, we need the NTL opening time, $t^{\mathrm{ntl}}_{\mathrm{open}}$, and closing time, $t^{\mathrm{ntl}}_{\mathrm{close}}$. They depend on the NTL clock latency ($t^{\mathrm{ntl}}_{\mathrm{del}}$), from PPW and from T. Since it is difficult to know $t^{\mathrm{ntl}}_{\mathrm{del}}$ at this stage, we assume that it is equal to $t^{\mathrm{HCP}}_{\mathrm{capture}}$, unless an NTL is merged into a NETF when we use the latency $t^{\mathrm{HCP}}_{\mathrm{launch}}$.

  In Fig. 3.4, the NTL would have the clock latency of PTL X. Lines 4–10 implement these computations. The condition on line 4 checks if the pin is the output of a PTL, thus the resulting NTL would be merged into a NETF.

- The additional delay from NTL insertion is ignored because it is usually small compared to the SCP_p delay and because it is hard to estimate before the layout. Note that we ignore it only to simplify the TG generation, but in the final layout step the P&R tool does consider the NTL delays.

Algorithm 2 Estimated Setup Slack attribute for pin p

Inputs: Parameters from Table 3.1 **Output:** ESS_p

1: **if** $HS_p \geq 0$ **then**
2: $ESS_p \leftarrow \infty$
3: **else**
4: **if** p is output of PTL **then**
5: $t^{\text{ntl}}_{\text{del}} \leftarrow t^{\text{HCP}}_{\text{launch}}$
6: **else**
7: $t^{\text{ntl}}_{\text{del}} \leftarrow t^{\text{HCP}}_{\text{capture}}$
8: **end if**
9: $t^{\text{ntl}}_{\text{open}} \leftarrow t^{\text{ntl}}_{\text{del}} + PPW$
10: $t^{\text{ntl}}_{\text{close}} \leftarrow T + t^{\text{ntl}}_{\text{del}}$
11: **if** $AT^{\max}_p < t^{\text{ntl}}_{\text{open}}$ **then**
12: $ESS_p \leftarrow SS_p - t^{\text{ntl}}_{\text{open}} + AT^{\max}_p + mTB_{\text{end}}$
13: **else**
14: **if** $AT^{\max}_p > t^{\text{ntl}}_{\text{close}}$ **then**
15: $ESS_p \leftarrow t^{\text{ntl}}_{\text{close}} - AT^{\max}_p$
16: **else**
17: $ESS_p \leftarrow SS_p + mTB_{\text{end}}$
18: **end if**
19: **end if**
20: **end if**

We explain the steps in Algorithm 2 analyzing the four cases which cover all the possible combinations, while in Fig. 3.3a we illustrate an example of TG for the circuit of Fig. 3.1b:

Case 1 — Positive hold slack If the considered p has positive hold slack, HS_p, then there is no violation to fix. To reduce the number of NTLs that will be used after retiming, all the NTLs that would be placed close to pins not belonging to short paths will not be added to the PTL netlist. Avoiding NTL insertion means no SS_p degradation, hence in this case we set weight (W) to ∞ (lines 6–7 of Algorithm 2). An example is the edge $D1O \rightarrow N3I$ in Fig. 3.3a, corresponding to the edge $D1 \rightarrow N3$ in Fig. 3.1b, which does not belong to a short path.

Case 2 — NTL close to the source PTL The additional delay caused by the late opening of the NTL may cause a setup violation, as shown in Fig. 3.1c. Attribute computation estimates the degradation of the pin setup slack, taking into account the late arrival time at the selected pin ($AT^{\max}_p$), SS_p, $t^{\text{ntl}}_{\text{open}}$, and the margin for time

borrowing (mTB_{end}). The delay introduced by the NTL can be tolerated up to mTB_{end}. Lines 11–12 of Algorithm 2 perform these computations. An example is edge $AQ \rightarrow N1I$ from Fig. 3.3a, corresponding to the edge from PTL A to $N1$ in Fig. 3.1c.

Considering that the SCP for this edge ends in PTL Z, the parameters SS_p, $AT_p^{\max}$ and mTB_{end} are all equal to 0 because the SCP delay is equal to the T added to the maximum time borrowing. $t_{\text{open}}^{\text{ntl}}$ is equal to 2 for all the cases shown in Fig. 3.3a because the clock is considered ideal. Given the previous considerations, compute ESS_p:

$$ESS_p = 0 - 2 + 0 + 0 = -2 \tag{3.2}$$

Case 3 — NTL close to the sink PTL If the input signal of the sink PTL belongs to a critical path, then the setup constraints added by the early NTL closing will likely prevent satisfying the setup constraints. If the late arrival time at the pin, $AT_p^{\max}$, exceeds $t_{\text{close}}^{\text{ntl}}$, then the signal will not pass through the NTL. The SS_p degradation is computed as the difference between these two values (lines 14–15 of Algorithm 2). An example is edge $D2O \rightarrow ZD$ from Fig. 3.3a, corresponding to the edge from $D2$ to PTL Z in Fig. 3.1d. $t_{\text{close}}^{\text{ntl}}$ is equal to 4 for all the cases because the clock is considered ideal and $AT_p^{\max}$ is 6. Given the previous considerations, ESS_p is computed as:

$$ESS_p = 4 - 6 = -2 \tag{3.3}$$

Case 4 — General case If none of the previous cases occurs, then $AT_p^{\max}$ at the NTL input falls into the NTL transparency interval and there is no SS_p degradation (line 17 of Algorithm 2). An example is edge $D1O \rightarrow YD$ from Fig. 3.3a, corresponding to the edge from $D1$ to PTL Y in Fig. 3.1d.

Considering that the SCP for this edge ends in PTL Y, $SS_p = 1$ because the signal arrives 1 time unit before the rising edge of the clock, while $mTB_{\text{end}} = 2$ because there is no time borrowing. Given the previous considerations, ESS_p can be computed as:

$$ESS_p = 1 + 2 = 3 \tag{3.4}$$

Sub-path delays

The second value of the tuple, D^p_{ptl}, shows the delay of the path between the pin p and the endpoint PTL of SCP_p. It is equal to the difference between $AT^{\max}_{\text{end}}$ and $AT^{\max}_p$

$$D^p_{\text{ptl}} = AT^{\max}_{\text{end}} - AT^{\max}_p. \tag{3.5}$$

The third value of the tuple, D^{ptl}_p, shows the delay of the path between the start point PTL of the SCP_p and the pin p. It is computed as the difference between $t^{\text{SCP}}_{\text{launch}}$ and $AT^{\max}_p$

$$D^{\text{ptl}}_p = AT^{\max}_p - t^{\text{SCP}}_{\text{launch}}. \tag{3.6}$$

3.1.5 Short-Path Graph

The Short-Path Graph (short-path subgraph (SPG)) is a subgraph of the TG that only contains the pins and edges that belong to short paths, i.e., all those pins p such that $HS_p < 0$. Hold violations will be fixed by finding a cut (subset of edges) of the SPG where the NTLs will be inserted.

Two types of edges can be distinguished in the SPG:

$$E = E_{\text{cells}} \cup E_{\text{wires}}$$

whereas E_{cells} correspond to those edges that connect input-to-output pins in combinational cells and E_{wires} correspond to the remaining edges. The cut of the SPG must be defined using edges in E_{wires}.

The insertion of an NTL in an edge may benefit from the presence of an adjacent PTL at the start or end point of the edge. Thus, both latches can be merged into an FF, either PETF (NTL-PTL) or NETF (PTL-NTL), as shown in Fig. 3.1f. Thus, we can define

$$E_{\text{wires}} = E_{\text{ff}} \cup E_{\text{lat}}$$

to distinguish these edges, with E_{ff} representing the edges in which the merging is possible and E_{lat} representing the remaining edges. Additionally, two parameters are defined to represent the cost of inserting an NTL, c_{ff} and c_{lat}, with $c_{\text{ff}} < c_{\text{lat}}$, since merging implies area savings. For the considered technology, when comparing

latches and flip-flops with lower fanouts, the ratio of the area $\frac{A_{lat}}{A_{ff}}$ is approximately 50 %. However, when considering cells with higher fanouts, the ratio significantly increases, reaching around 75 % in the worst cases. Additionally, taking into account the extra cells required to route the clock tree of the additional sequential elements and the greater variety of cells available in the flip-flop library, merging appears to be a more favorable choice. These parameters can be tuned to control the area overhead of the solution.

Graph 3.3b shows the SPG of the example circuit from Fig. 3.1b.

Table 3.2 Variable definitions for Alg. 3. (ILP Model)

NAME	DEFINITION
SPG	Short-Path Graph
T	Cycle period
δ	Fraction of T to meet setup constraints
c_{ff}	Cost of merging an NTL with a PTL
c_{lat}	Cost of not merging an NTL ($c_{\text{lat}} > c_{\text{ff}}$)
E_{cells}	Edges between pins of the same cell
E_{wires}	Edges between pins of different cells
E_{ff}	Edges where NTL would be merged in PETF/NETF
E_{lat}	Edges where NTL would not be merged
$R(e,X)$	Edge selection value

3.1.6 Integer Linear Programming model

Starting from the SPG and the attributes computed from static timing analysis of the PTL post-layout netlist, an ILP model is defined to fix the hold violations and select the NTL locations. Alg. 3 and Tab. 3.2 show the ILP model and the definition of the algorithm variables.

For each pin (p) of the SPG, a binary variable p is created. For each edge (e), $p_{\text{end}}(e)$ and $p_{\text{start}}(e)$ represent the variables associated to the endpoint and the start point of e, respectively. Each edge is characterized by the edge selection value, $R(e,X)$, defined as

$$R(e,X) = p_{\text{end}}(e) - p_{\text{start}}(e). \tag{3.13}$$

Algorithm 3 Integer linear programming (ILP) Model

Inputs: SPG, T, δ, c_{ff}, c_{lat}
Output: Location of the NTLs (edges with $R(e) = 1$)

1: $E \leftarrow \text{Edges}(SPG)$
2: $E_{\text{cells}}, E_{\text{wires}}, E_{\text{ff}}, E_{\text{lat}} \leftarrow E$
3: $ESS_p, D_p^{\text{ptl}}, D_{\text{ptl}}^p \leftarrow \text{TimingAttributes}(E)$

$$\text{minimize} \quad c_{\text{lat}} \sum_{\forall e \in E_{\text{lat}}} R(e) \quad + \quad c_{\text{ff}} \sum_{\forall e \in E_{\text{ff}}} R(e) \tag{3.7}$$

$$\text{subject to} \quad \forall e \in E_{\text{cells}} : \quad R(e) = 0 \tag{3.8}$$

$$\forall e \in E_{\text{wires}} : \quad R(e) \geq 0 \tag{3.9}$$

$$\forall e \in E_{\text{wires}} : \quad R(e) \cdot ESS_p(e) \geq 0 \tag{3.10}$$

$$\forall e \in E_{\text{wires}} : \quad R(e) \cdot D_{\text{ptl}}^p(e) \leq \delta \cdot T \tag{3.11}$$

$$\forall e \in E_{\text{wires}} : \quad R(e) \cdot D_p^{\text{ptl}}(e) \leq \delta \cdot T \tag{3.12}$$

The cut (location of the NTLs) is defined for those edges with $R(e,X) = 1$, i.e., $p_{\text{start}}(e) = 0$ and $p_{\text{end}}(e) = 1$, as shown in Fig. 3.6.

The cost function (3.7) accounts for the number of new sequential elements added to the circuit, i.e., the number of NTLs inserted in edges not connected to a PTL. This will push the solution of Algorithm 3 to use as many NETFs and PETFs as possible to reduce the final number of sequential elements in the circuit.

The constraint (3.8) avoids that Alg.3 selects edges representing connections between pins of the same cells (E_{cells}).

The constraint (3.9) enforces $p_{\text{end}}(e) \geq p_{\text{start}}(e)$, because $p_{\text{end}}(e)$ and $p_{\text{start}}(e)$ are binary this restricts $R(e,X)$ to be binary. It also implies that all pins p belonging to a path that reaches $p_{\text{start}}(e)$ will have $p = 0$, while all pins belonging to a path that crosses $p_{\text{end}}(e)$, reaches p, and ends at a PTL will have $p = 1$. Then, the algorithm splits the graph in two partitions, before and after the NTLs, by removing the edges with $R(e,X) = 1$. The partition in which all pins have $p = 1$, i.e. the part of the graph that includes the PTL endpoints, will have no early arriving signals. Fig. 3.6 shows an example of the graph partitioning generated by the model.

Although solving an ILP generally has very high runtime, in this particular case it is very close to a max-flow min-cut problem, which is known to have polynomial complexity. This is the likely reason why the runtime of our algorithm remains very small, as shown in Table 4.2, even for designs with tens of thousands of gates and

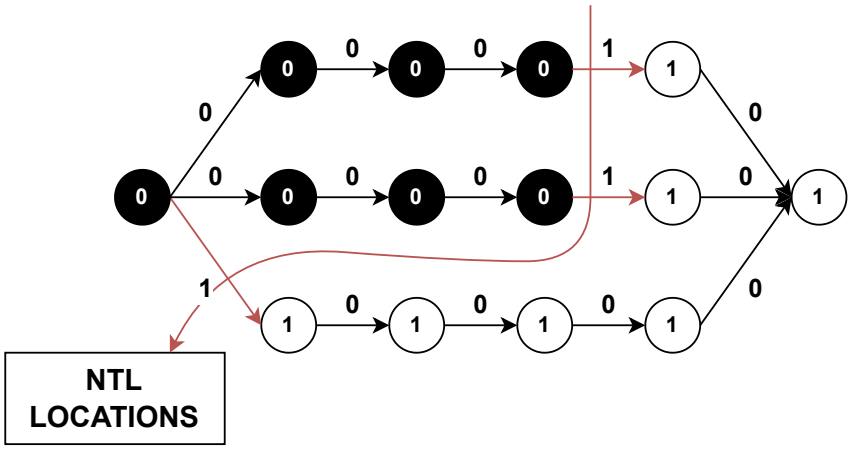
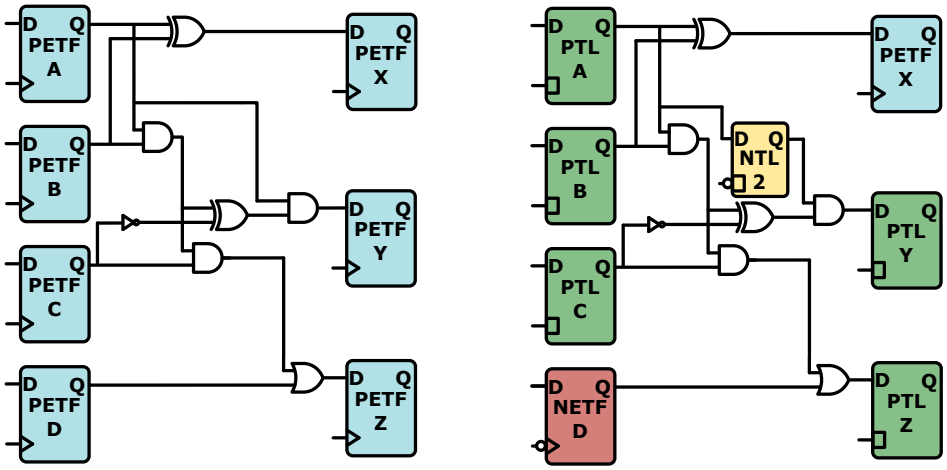


Fig. 3.6 Solution example showing the cut chosen for the SPG from Fig. 3.3b. The vertex attributes correspond to the P variables of the ILP model and the edge attributes represent the edge selection value ($R(e,X)$) computed for each edge. The vertices with input edge attribute equal to 1 are selected for NTL insertion.

FFs. The development of a heuristic algorithm is left to future work, if the execution time becomes excessive, e.g. comparable to or larger than the physical design time.

The constraint (3.10) guides the model towards solutions that do not worsen setup violations, because the SS_p for each selected edge for NTL insertion must be greater than zero. The purpose of this constraint is to prevent the insertion of a NTL at a location that would introduce delay to a setup critical path. The estimation done in ESS_p is an approximation of the final SS_p that takes into account not only the length of the combinational logic delay, but also the clock tree latency generated by the layout tool, as discussed in Section 3.1.4.

However, this is an approximation and we need two more inequalities, (3.11,3.12), to simplify the problem of meeting the setup constraints. The D^p_{ptl} and D^{ptl}_p attributes report the distance, in terms of post-layout delay, between each pin p and the source/sink PTLs. An NTL placed in front of p divides the path in two parts and the two graphs give an estimation of the length of these sub-paths. To make these paths as short as possible, these time intervals are constrained to be a fraction δ of T, that is a parameter of our algorithm. The value of δ, with $0 < \delta < 1$, is discussed in the next section.



(a) Generic PETF-based circuit (b) Resource merging

Fig. 3.7 Example of *Mix & Latch* optimization on a PETFs-based circuit (Fig. 3.7a). Merge sequences of complementary latches: PTL-to-NTL into NETF, NTL-to-PTL into PETF (Fig. 3.7b). Combinational gates have unit delay only for ease of explanation. We consider as short paths those that cross one gate.

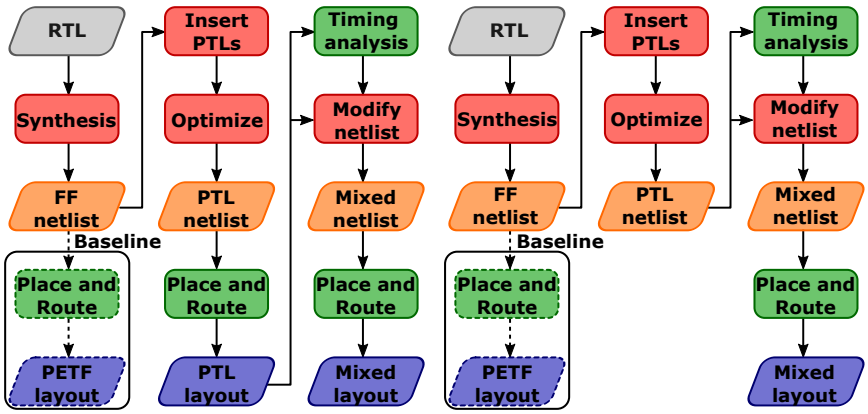
3.2 Updated flow

In this section, we aim to further increase performance with respect to Section 3.1, reaching a level that is at least comparable to retiming and avoiding resource increase.

Our main contributions to the state-of-the-art are:

1. Speed up the original *Mix & Latch* flow by avoiding one P&R iteration.
2. Relax the timing analysis both in terms of setup constraints, which control the performance of the circuit after NTL insertion, and of how to select which hold violations the NTL insertion will solve.
3. Propose a flow considering only setup critical paths for PTL insertion.

Fig. 3.7 shows the *Mix & Latch* methodology using a simplified example. Starting from a FF-based netlist (Fig. 3.7a), all registers are replaced by PTLs to fully exploit time borrowing, but at the risk of hold time violations. To solve them, *Mix & Latch* inserts NTLs on the short paths that are at risk. Such latches act as retention barriers, delaying any signal that travels through a short path. Choosing the best location for these latches is not trivial. In fact, if placed incorrectly, these NTLs lead to setup



(a) Original flow (b) New flow

Fig. 3.8 Comparison between the original (Fig. 3.8a) and the new post-synthesis (Fig. 3.8b) implementation flow. The new flow performs timing information extraction and netlist manipulation after the PTL-based netlist synthesis.

violations. Additionally, the latches can be merged if there is a direct PTL-to-NTL or NTL-to-PTL connection, resulting in either a PETF or a NETF, as shown in Fig. 3.7b.

Mix & Latch uses an ILP model to compute a cut from the PTL post-layout circuit graph determining the best NTLs position. The ILP model aims to minimize the new sequential elements (NTLs) added to the circuit, resolve hold violations, and avoid performance degradation.

Three timing constraints limit the potential timing degradation produced by the insertion of NTLs in critical paths and help satisfy the setup constraints. The cost function optimizes the area overhead. The ILP model is similar to a max-flow formulation, and the runtime has proven to be manageable even for large designs.

Fig. 3.8 shows a comparison between the original flow and our new proposal for the *Mix & Latch* methodology.

To overcome the limitations of the original *Mix & Latch* flow, we introduce two innovations: (1) pessimism reduction and (2) post-synthesis extraction of timing information. The rest of this section analyses the proposed improvements.

3.2.1 Post Synthesis Flow

The original *Mix & Latch* flow has typically long execution times because it requires two P&R steps and derives the TG taking into account the delay contributions from the interconnections. However, since it only *estimates* the best positions for NTLs, it cannot guarantee that the solution will produce the desired results, since cell placement and routing is left to the P&R tool, and the resulting netlist changes cannot be handled by the tool in incremental mode (also known as ECO mode). To address both problems, a so-called *post-synthesis* flow is introduced, where the TG and cut computations are performed on the netlist after the synthesis step, as shown in Fig. 3.8, instead of after the P&R of the PTL-based netlist.

3.2.2 Reduce Pessimism

As shown in Fig. 3.8a, the original *Mix & Latch* methodology uses post-P&R information to estimate the effect of the NTL insertion. However, since the netlist changes caused by NTL and NETF are too large to be handled by incremental implementation, the estimate of the impact of their insertion performed by *Mix & Latch* is necessarily imprecise.

For setup time, some promising solutions may be discarded due to small negative values of the estimated setup slack after NTL insertion, ESS_p, which poses a hard exclusive constraint to the solver and instead may be fixed with P&R. For hold time, it may not be necessary to resolve all violations via NTLs, as small ones can be resolved by the P&R tool by inserting buffers and/or resizing cells. To address these issues, two parameters are introduced to reduce the pessimism of the algorithm:

- Max setup derate *MSD*;
- Max hold derate *MHD*.

MSD is multiplied by the clock period T and added at the end of Algorithm 2, where the final line becomes:

$$ESS_p \leftarrow ESS_p + T \times MSD. \tag{3.14}$$

In this way, small negative values of ESS_p become positive, thus avoiding to discard a potentially viable solution. As discussed in Section 4.2, small values of this parameter (e.g., 10 % of T) are sufficient to achieve better performance.

While *MSD* tries to increase the number of possible solutions, *MHD* reduces the number of paths included in the short-path subgraph, leaving the solution of some hold violations to the P&R tool. Hold violations that have an absolute value smaller than $T \times MHD$ will not be addressed through NTL insertion. Instead, they will be resolved during the implementation process using conventional methods like buffer insertion. The results show that small values of this parameter (e.g., also 10 % of T) also help to achieve better performance, while too high values would lead to excessive hold violations that cannot be fixed in the final layout.

3.2.3 Worst-Path Flow

Another strategy for reducing the complexity of the optimization problem is to only take into account the flip-flops that are endpoints of the setup critical paths rather than converting all of them to PTLs. This is called *worst-path* flow and use the Worst Setup Slack (WSS) parameter. Only flip-flops with setup slack less than WSS at the input data pin are taken into account. Experimental results show that the optimal value for this parameter can change based on the target frequency. For instance, both substituting the majority of FFs into latches and substituting a small percentage of them can lead to a viable solution of the *Mix & Latch* algorithm. Future work will concentrate on finding a correlation between WSS values and final area and power results to automatically predict the best configuration of this hyperparameter.

3.2.4 Formal verification

[35] shows that keeping each register in its original position, and possibly adding registers that will be ignored in formal verification, as done in *Mix & Latch*, simplifies formal verification. The equivalence of circuit behavior involves a combination of Logic Equivalence Checking (LEC) and STA. STA guarantees that modifying the characteristics of registers does not introduce data sampling errors compared to the original FF-based netlist, and is fully discussed in [36]. LEC ensures that the introduction of retention barriers (NTLs) in the circuit does not alter the behavior of the combinational logic. During LEC, the NTLs operate in transparent mode, setting their input clock to 0. Differently from *retiming*, this methodology does not require annotations from the synthesis step.

Chapter 4

Experimental Results

4.1 Original flow

Open-source `PULP` [37] library is used to model the ILP, the default solver is `CBC`. To evaluate the proposed algorithm, we apply the optimization flow to 13 circuits from a pool of benchmarks, each implemented at a range of operating frequencies. Four circuits are cryptographic IPs from the CEP benchmark [38], eight are from the ISCAS89 benchmark [39], and one is a small processor core from the ITC99 benchmark [40]. The implementation flow uses an industrial 28 nm FDSOI CMOS technology, Design Compiler from Synopsys for logic synthesis, and Innovus from Cadence for P&R.

We set $\delta = 0.75$ in Algorithm 3, i.e. the maximum sub-path delay is 75 % of T. Since δ defines the length of the sub-paths generated by NTL insertion, 75 % for a *DC* of 50 % means that the two sub-paths are reasonably balanced. Further exploration of the impact of δ is left to future work.

We also set $c_{\text{ff}} = 0$ and $c_{\text{lat}} = 1$ to account for the number of new sequential elements in the circuit.

Table 4.1 shows the frequency improvement for the considered benchmarks, together with the final sequential resource mix. The average improvement in frequency is about 1.33X. We used a granularity of 0.1 ns in the exploration of the minimum clock period (T). The algorithm is doing better than average for the cryptography

Table 4.1 Operating frequency and sequential resources for designs from ISCAS$^{\diamond}$, CEP$^{\circ}$ and ITC99$^{\bullet}$ benchmarks. Columns labeled 'Original" refer to PETF-based layouts, while those labeled "mixed" refer to the optimized ones.

Design	f_{max} (GHz)			Original	Mixed			
	Original	Mixed	Ratio	*PETF*	*PETF*	*NETF*	*PTL*	*NTL*
s1196$^{\diamond}$	2.50	3.33	1.33	18	1	0	17	19
s1423$^{\diamond}$	2.00	2.50	1.25	74	5	4	65	128
s5378$^{\diamond}$	1.67	2.00	1.20	176	80	3	93	105
s9234$^{\diamond}$	2.00	2.50	1.25	145	47	28	70	113
s13207$^{\diamond}$	1.00	1.43	1.43	625	395	137	93	521
s15850$^{\diamond}$	1.25	1.67	1.33	442	94	88	260	453
s38417$^{\diamond}$	0.48	0.67	1.40	1564	690	110	764	1213
s38584$^{\diamond}$	0.43	0.53	1.21	1275	636	136	503	1116
b22$^{\bullet}$	0.48	0.67	1.40	613	78	72	463	1141
des3°	0.67	1.00	1.50	199	34	65	100	125
md5°	0.43	0.67	1.53	269	71	61	137	519
sha256°	0.56	0.62	1.12	1040	502	284	254	579
aes_192°	$*$[1]	0.33	$*$[1]	9382	0	9153	229	530

IPs like des3 and md5, probably because they are designs with acyclic paths that are generally not well-balanced.

Fig. 4.2 shows the frequency improvement and the area comparison considering the maximum frequency for the original design and the optimized one. In most of the cases there is an area penalty which can exceed 1.2 X. However, this is compensated by a maximum frequency increase above 1.2 X for these designs. There are also cases in which the performance improvements do not cause any area increase, like for des3, sha256, s38584 and b22.

Fig. 4.1 shows the results obtained at frequencies at which both design versions meet the timing for a meaningful area comparison. To demonstrate the actual scalability of this approach, Table 4.2 shows the runtime of the ILP algorithm compared to the time needed for the layout in the three cases. The ILP runtimes are always less than 10% of the layout times.

[1]The maximum frequency reached for the original designs is low compared to [14] and to the mixed result. For this reason, we do not report it for the frequency and runtime comparisons.

Table 4.2 ILP execution time (s) and layout times (s). Orig layout refers to the starting PETF netlist, PTL layout to the netlist without hold constraints and with only PTLs, and mixed layout to the final step after NTL insertion. The columns #SEQ. and #COMB. report the number of sequential and combinational elements in the PTL layout, which is the netlist analyzed and provided to the ILP solver.

Design	Orig layout (s)	PTL layout (s)	#SEQ.	#COMB.	ILP (s)	Mixed layout (s)
s1196	461	425	18	332	1	461
s1423	698	577	74	456	1	670
s5378	1078	925	176	645	5	1219
s9234	620	554	145	503	2	566
s13207	4867	4151	625	1918	11	5049
s15850	816	784	442	1589	6	924
s38417	1252	1651	1564	4049	24	5155
s38584	1624	1809	1275	8058	32	1506
b22	2044	2306	613	11026	29	2242
des3	727	746	199	1795	7	725
md5	4175	3838	269	10639	19	2114
sha256	3456	2406	1040	4116	30	3121
aes_192	$*$[1]	39703	9382	130264	3130	44800

4.1.1 Timing closure

The P&R tool converges to a good solution if, at the end of the automated implementation flow, the hold and setup violations are small and can be fixed with only a few iterations of the final design optimization commands. If they are too large, then the designers typically conclude that the P&R tool cannot implement the design at that specific frequency. In these cases we do not report the area because it is usually excessive. We do at most five optimization iterations to solve the remaining setup and hold violations.

4.1.2 Area comparison

Fig. 4.1a shows the area comparison at different frequencies in the cases in which both the FF-based and the mixed designs meet the timing. The optimized designs from the ITC99 and CEP benchmarks also have a smaller area than the original ones. However, this is not true in general for the circuits from the ISCAS89 benchmark.

Fig. 4.1b shows the same area comparison as Fig. 4.1a, but in this case the x-axis shows the ratio of FFs in the mixed design compared to the original netlist

$$\frac{FF\_MIXED}{FF\_ORIG} = \frac{\max(\#PETF_{\text{MIXED}}, \#NETF_{\text{MIXED}})}{\#PETF_{\text{ORIG}}} \tag{4.1}$$

where $\#PETF_{\text{ORIG}}$ is the number of PETF in the original circuit. The paths that constrain the design the most are those between pairs of same polarity FFs, because paths from PETF to NETF allow time borrowing and paths from NETF to PETF cannot be generated by Algorithm 3. This is why in (4.1) we consider the maximum between the two FF types, rather than the sum.

Fig. 4.1b shows that the area increase in the mixed designs is well correlated with the ratio FF_MIXED/FF_ORIG.

In some circuits, even our cost function, which drives the solution to use as many FFs as possible, could lead to considerable overhead of the mixed version area. Fig. 4.1c and Fig. 4.1d show the sequential and combinational area comparison. The sequential area increases in most examples because of the higher number of sequential elements in the design. However, for the designs with a low FFs ratio, easier timing convergence reduces the number of high speed gates.

Thus, it tends to compensate this overhead and sometimes leads to a smaller total area. In the next section, we analyze the effect on the area overhead of modifying the NETF allocation cost in the ILP model. We show that results in a significant improvement in the worst cases. We conjecture that power would also be improved, but its evaluation is outside the scope of this paper, which focuses on *performance gains* with limited area cost, or even with area improvement.

4.1.3 Algorithm tuning to reduce area overhead

To reduce the area overhead, we discouraged the use of NETFs by increasing the cost of inserting NTLs in locations enabling the PTL-NTL merging.

We slightly modified the ILP model by defining a different cost for merging latches into NETF (cost 1) or PETF (cost 0). Fig. 4.1e shows that the original area overhead for the ISCAS89 circuits is reduced.

Although this configuration improves the quality of the ISCAS89 worst cases, it increases the area compared to Fig. 4.1a for some of the `des3`, `md5`, and `b22` designs. Considering the best result among these two values for NETF costs, the

average area improvement is 1.19 X over the considered benchmarks, with above-average performance for the cryptography IPs. In some cases belonging to ISCAS89 benchmark, the area increases considerably. Addressing this issue, e.g. by further tuning the algorithm parameters, is left to future work.

4.1.4 Comparison with other work

Some of our results can be directly compared with those presented in [14], which converts an FF-based netlist to a 3-phase PTL-based netlist using two variants of the same algorithm. While our main goal is to improve the maximum operating frequency, [14] focuses instead on reducing the area occupation and power consumption. This study demonstrates that utilizing latches and relative retiming techniques can result in an average reduction in power consumption of over 20 %. Despite the differences in technology, implementation setup and target optimization, the area overhead introduced by our optimization algorithm is compared with the results of [14]. There are six common benchmark circuits used by us and [14], four from CEP and two from ISCAS89. For the CEP benchmarks, [14] reports maximum area reductions at 500 MHz of 14 % for `des3`, 17.7 % for `sha256`, and 5.8 % for `md5`. Our results in Fig. 4.1d and Fig. 4.1e show that, for the cryptography IPs our area reduction exceeds [14], with peaks of 22.38 % for `des3`, 41.32 % for `sha256`, and 51.13 % for `md5`. But for most ISCAS89 benchmark circuits our algorithm increases or only slightly reduces the area, while the area reduction achieved by [14] is more than 10 %. Specifically, our algorithm reduces the area by 5.29 % for `s1423`, 7.59 % for `s5378`, and 8.82 % for `s38584`, and increases the area by 3.28 % for `s38417`, 9.35 % for `s9234`, 4.79 % for `s1196`, and 41.75 % for `s13207`. Note that performance, which is our main design goal, is improved in all cases.

4.2 Updated flow

4.2.1 Test Setup

We selected a *Zero-riscy* core [41] to be implemented using the above flows because RISC-V cores have become increasingly popular in recent years, and because processors contain both acyclic and cyclic paths, e.g., in arithmetic units and FSMs

respectively. This variety of subcircuit topologies helps us to test the *Mix & Latch* methodology under stringent conditions, since the original flow Section 4.1 showed that it is most effective on acyclic circuits. To perform the power analysis with realistic switching activity information, the standard delay format (SDF) simulation step is performed with backannotated delays obtained while running an advanced encryption standard (AES) algorithm on the RISC-V core. At the end of the test, the flow compares the values stored in the data memory with those obtained during a reference RTL simulation to ensure the correctness of the result. This increases confidence in the correctness of the implementation because it validates the timing constraints added to the backend flow.

The RISC-V core is synthesized with Synopsys Design Compiler, which also performs the retiming optimization used for comparison. Physical synthesis is then performed using Cadence Innovus. As mentioned in the Section 3.2, we use a 28 nm *CMOS FD-SOI* technology.

From a sweep of their value from 0 % to 20 %, the best value obtained for both *MSD* and *MHD* (discussed in Section 3.2.2) is 10 % of the clock period for the new *Mix & Latch* flows.

Since the cell library does not provide NETFs cells, we used PETFs cells with an inverter on the clock pin. Thus, replacing a PTL-NTL pair with a NETF to increase frequency penalizes the design density due to the additional inverters, and we disabled it during the experiments.

The *worst path* variation is applied on top of the *post-synthesis* flow. To adjust the WSS threshold for the worst-path flow, the setup slack of the critical path is used as a starting point and a sweep is performed in 50 ps steps until the clock period value. Then is selected the WSS value which delivers the best area and power performance after achieving STA closure. Automatic tuning of this parameter is left for future work.

4.2.2 Performance, Area, and Power Analysis

For each flow, a clock period constraint sweep is performed to find the maximum operating frequency. Table 4.3 shows the performance results for minimum clock period T_{min}, maximum frequency f_{max} and relative frequency gain f_{gain} compared to the baseline FF-based implementation.

Table 4.3 Performance comparison of implementation flows

Implementation flow	T_{min} (ns)	f_{max} (MHz)	f_{gain}
FF without retiming (baseline)	3.00	333.3	1.0 ×
Mix & Latch original	2.50	400.0	1.20 ×
FF with retiming	2.15	465.0	1.40 ×
Mix & Latch post-syn	2.00	500.0	1.50 ×
Mix & Latch worst-paths	2.00	500.0	1.50 ×

Compared to baseline implementation, both the post-syn flow and the worst-path version of *Mix & Latch* flow improve the frequency over classical retiming. This result, obtained on a more complex benchmark than Section 4.1, is very promising for the *Mix & Latch* methodology, implying that it may become a viable alternative to logic retiming in industrial flows.

Note that the original *Mix & Latch* flow applied to this design failed, because it added too many NTLs, which eventually caused timing violations in the mixed layout. The 400 MHz clock frequency was achieved only by using the *MSD* and *MHD* parameters introduced in this paper. This suggests that the choice of best values for the *Mix & Latch* parameters (like many other parameters in modern physical implementation flows) may vary with design characteristics. For example, the Zero-riscy RISC-V core may have more short paths than the previously considered benchmarks, and relaxing the timing constraints of *Mix & Latch* seems an effective strategy to address this issue.

Fig. 4.3 and Fig. 4.4 plot power, area, and cell usage for each implementation flow versus clock frequency, highlighting their components. Missing points in the plots are caused by failed STA at the end of the selected flow. Note that the original *Mix & Latch* flow struggles the most to achieve timing closure, while the *post-synthesis* and *worst path* flows both provide a feasible solution for every target frequency, with the sole exception of the 500 MHz clock frequency for the *post-synthesis* flow.

In general, designs optimized using both the original and *post-synthesis Mix & Latch* flows consume more power than the retimed design, while their area is comparable or smaller, except at the highest frequencies, as shown in Fig. 4.3a. The *worst path* flow instead manages to deliver the smallest area occupation compared to

the other flows for each target frequency. Moreover, it also achieves smaller power consumption, apart from the highest retiming frequencies.

The area and power results show how the original *Mix & Latch* flow is suboptimal with respect to the newest improvements. This is probably because the *Mix & Latch* algorithm has trouble predicting the choices the P&R tool makes in the second layout step. Instead, the *post-synthesis* implementation delivers on average a power and area overhead of 11.10 % and 5.33 % , respectively, compared to the retiming flow. Finally, the *worst path* implementation achieves a power reduction of 5.53 % compared to retiming along with an area reduction of 12.63 % . Therefore, it is possible to use either the *post-synthesis* or *worst path* flow based on power, performance, and area (PPA) vs. design time constraints, since the first flow provides a viable solution at the first iteration, while the second requires some additional steps to fine-tune the WSS threshold.

4.2.3 Logic Cell Utilization

Comparing retiming with *Mix & Latch* results in Fig. 4.3a, it can be observed how using latches instead of FFs reduces the area occupied by sequential elements, even when the number of sequential cells is nearly doubled (see Fig. 4.3b). This is due to both latches being smaller than FFs and retiming resizing cells to improve performance, while we use the smallest PTL and NTL cells to further reduce area usage.

Retiming also seems to use more numerous and larger buffers compared to *post-syn* and *worst path Mix & Latch*, probably due to resizing like for FFs. Instead, the introduction of even a very small number of latches leads to a significant increase in inverter cells with respect to retiming. This result is explained by the stricter clock tree constraints used during clock tree synthesis when using both sequential cell types with different clock polarities. In our case, the P&R tool decided to instantiate more inverter cells, as shown in Fig. 4.3b, leading to a corresponding increase in inverter area in Fig. 4.3a.

It can also be observed how the results of the *Mix & Latch* flows can vary with timing constraints. For target frequencies from 455 MHz to 488 MHz, the *Mix & Latch* algorithms choose solutions in which 62.5 % of the original registers are kept as latches, while in most of the other observed cases, at *both lower and higher target*

frequencies, this percentage is less than 5 %. Since the final results highly depend on the hyper-parameters of the *Mix & Latch* algorithm, further research is required to obtain better predictions and avoid the manual tuning of the hyperparameters.

4.2.4 Power Contributions

Fig. 4.4 summarizes the power consumption estimates extracted from the back-annotated simulation. Fig. 4.4b shows the different power components: (1) switching power, consumed by charging and discharging the interconnect and load capacitances, (2) internal power, consumed by charging and discharging the internal gate capacitance and by short-circuit currents, and (3) leakage power. Fig. 4.4a shows power contributions broken down by type of logic elements: (1) clock tree cells, (2) combinational logic, and (3) sequential elements.

Switching power is the main component in Fig. 4.4b that determines the power overhead of the original and *post-synthesis Mix & Latch* implementations. This can be explained by the fact that PTLs and NTLs in the final design allow more glitches to propagate along logic paths. The *worst path* flow, on the other hand, manages to successfully reduce switching power relative to the other *Mix & Latch* flows, and also reduces this component relative to retiming at the lower target frequencies.

Another major contribution is clock tree elements, especially when most of the registers are ultimately kept as latches, as in the original *Mix & Latch* flow or the other two flows in the range from 455 MHz to 488 MHz, as discussed in Section 4.2.3. It is also possible to assess a positive correlation between the increase in inverter cells and clock power, further indicating how the *Mix & Latch* methodology increases the number of elements on the clock tree. The analysis of the necessary modifications to the clock gating for *Mix & Latch* is left to future work.

While some configurations of *Mix & Latch* significantly increase the clock power, these same configurations reduce internal and sequential power, both of which can be attributed to the reduction in the size of the sequential elements.

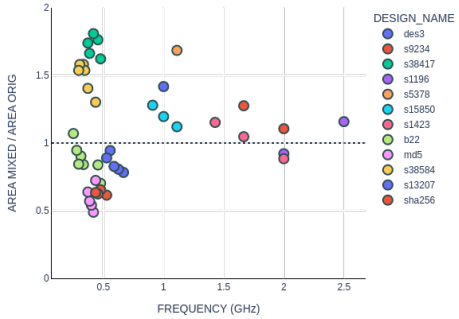
Static power increases for the original and *post-synthesis Mix & Latch* flows relative to the retimed implementation, while the *worst path* variant provides the lowest leakage power in most of its configurations.

Overall, the main term of the power graphs is the switching component, since Fig. 4.4b shows how the sum of all power components follows the same trend as the switching one. This suggests that by addressing glitch propagation and reducing the number of clock tree elements, it should be possible to minimize the power overhead of the *post-synthesis* implementation and achieve comparable values to the retiming implementation, while avoiding the multiple iterations required for the *worst path* flow.

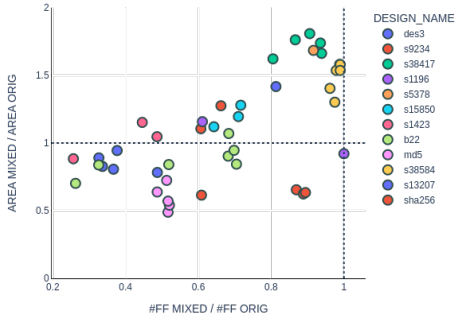
4.2.5 Mix & Latch With Retiming

Finally, we tried the new *Mix & Latch* flow *in conjunction* with retiming to further explore its potential. The P&R step only succeeds if retiming is performed either before TG extraction or after mixed netlist generation. Applying *Mix & Latch* to a retimed FF-based netlist or retiming after the PETF-to-PTL step results in a final layout that does not satisfy the timing constraints. In both cases, if P&R is successful, *there is no performance gain over the original* Mix & Latch *flow*, and in fact power and area usage increase compared to the results in Fig. 4.3 and Fig. 4.4. Although these results may be design dependent, they seem to imply that *the proposed* Mix & Latch *flow and retiming exploit essentially the same degrees of freedom in optimizing performance without changing the combinational logic*, and that no substantial gains can be achieved by combining them.

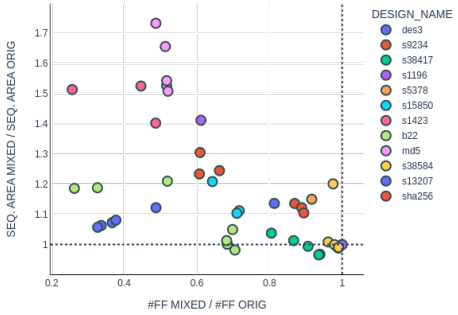
Thus, for the analyzed design, *Mix & Latch* is superior to retiming in terms of both maximum achievable performance and *compatibility with combinational equivalence checking*.



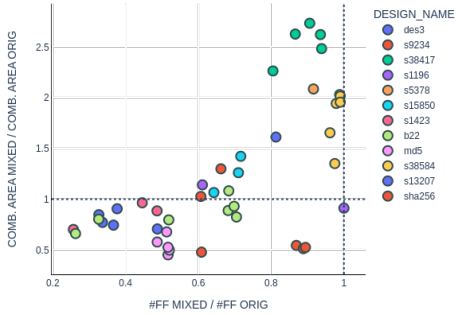
(a) Area vs. operating frequency



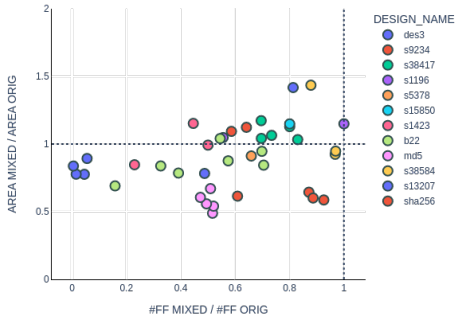
(b) Area vs. flip-flops used



(c) Sequential area vs. flip-flops used



(d) Combinational area vs. flip-flops used



(e) Area vs. flip-flops used with modified NETF cost

Fig. 4.1 Results of area comparison when both the mixed-based netlist and the PETF-based one successfully yield a layout

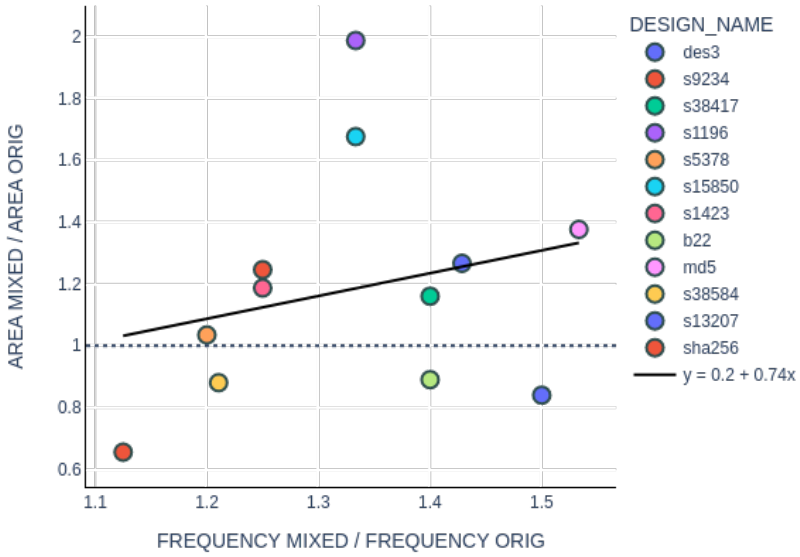
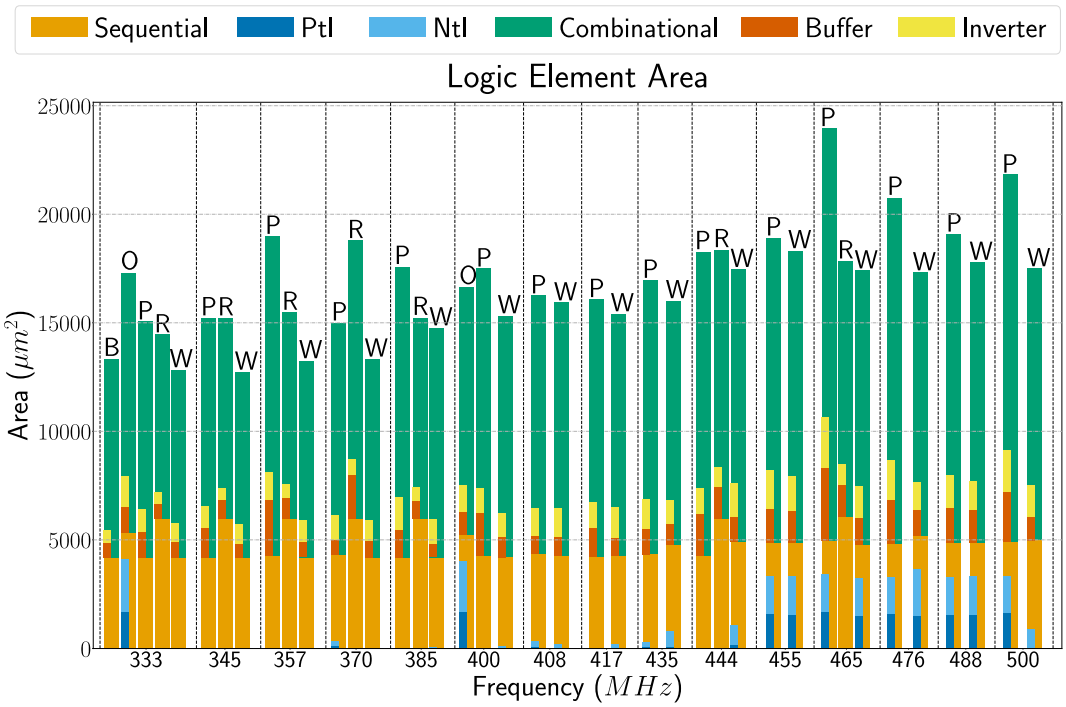
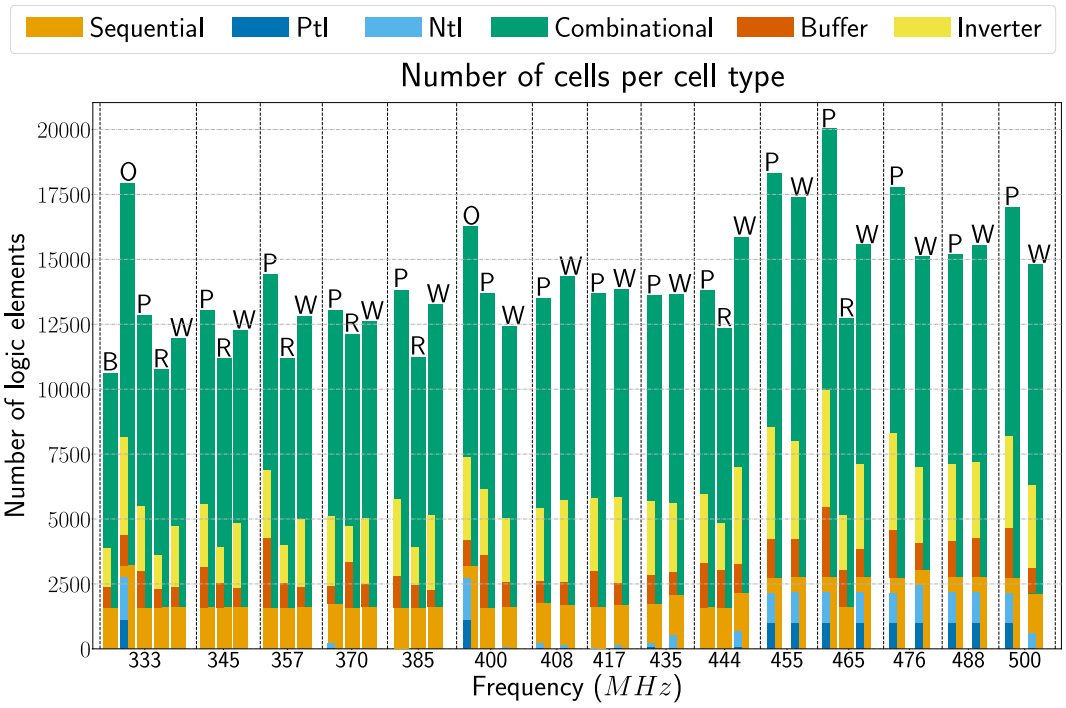


Fig. 4.2 Ratio of post-layout area, considering the layouts obtained at the highest working frequencies for both MIXED and ORIG versions, compared to the related frequency improvements. The black line shows the linear regression of the area increase with respect to the frequency gain. The offset and the slope of the line are stated in the legend.

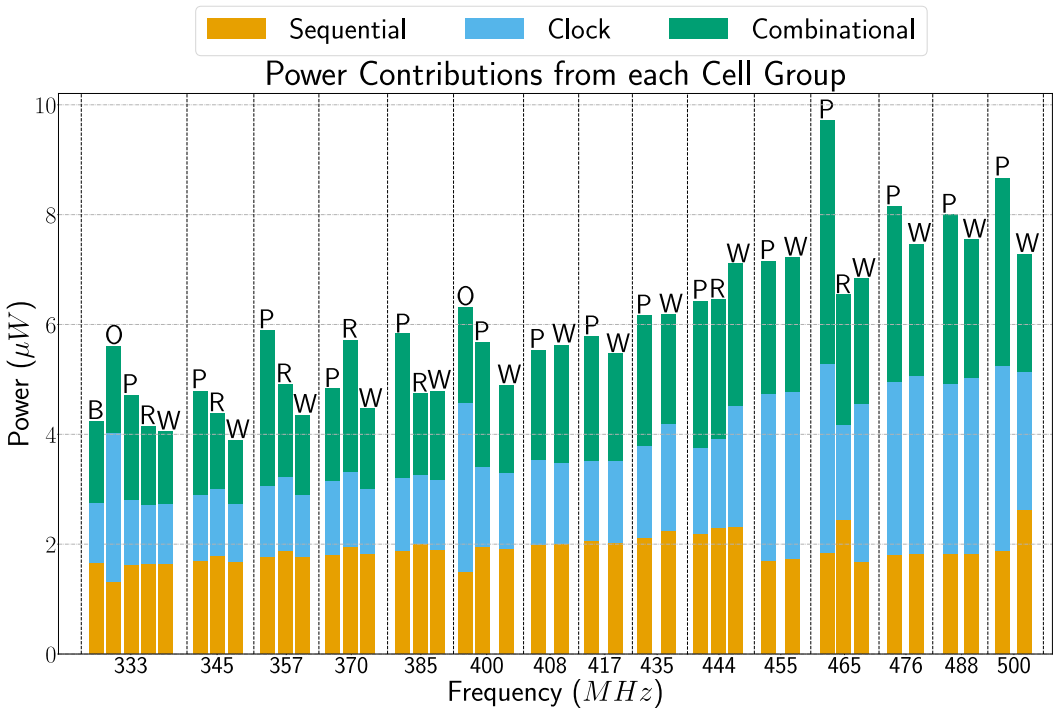


(a) Area of different logic cell types

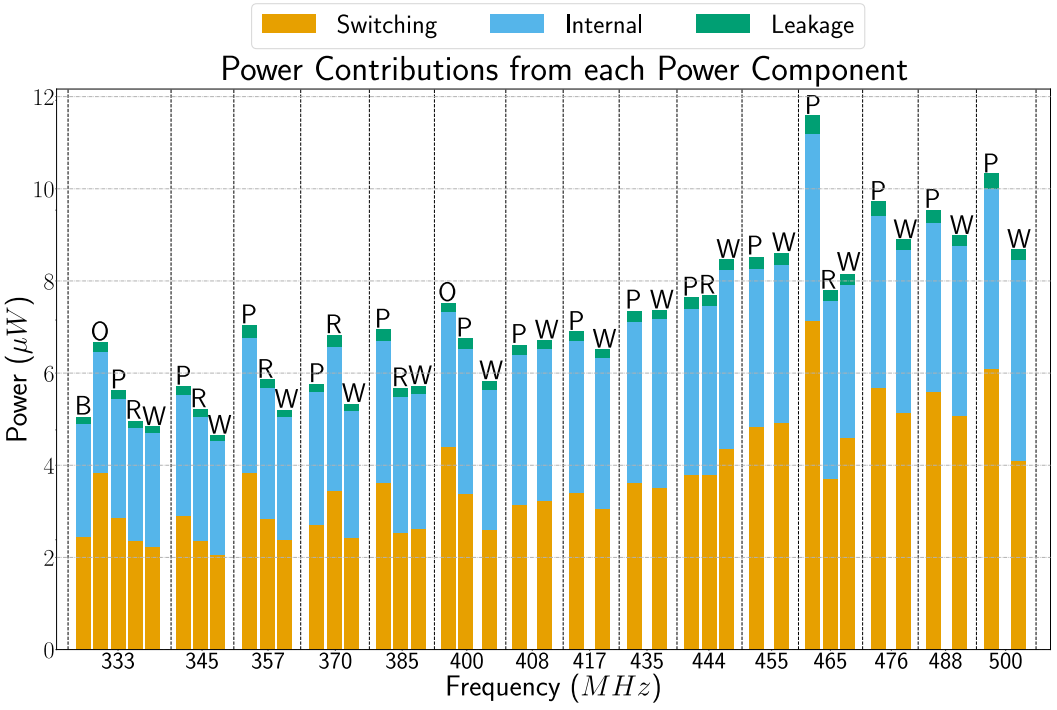


(b) Usage of different logic cell types

Fig. 4.3 Total area at different frequencies for baseline (B), original *Mix & Latch* flow (O), post-synthesis *Mix & Latch* (P), retiming (R), worst path (W), split into sequential and combinational contributions highlighting the fractions related to PTLs, NTLs, buffers, and inverters (Fig. 4.3a) and number of logic elements (Fig. 4.3b).



(a) Contribution of each cell group to the power consumption



(b) Internal, switching, and leakage power consumption

Fig. 4.4 Power consumption for original *Mix & Latch* (O), post-synthesis *Mix & Latch* (P), and retiming (R) at different operating frequencies, split by cell group (Fig. 4.4a) and by type of power (Fig. 4.4b).

Chapter 5

Conclusion and Future Work

This thesis introduces the *Mix & Latch* methodology that optimizes FF-based netlists by replacing the PETFs with PTLs, and solving the hold violations generated by such replacement using an efficient ILP model that selects a specific group of edges, and places NTLs on short paths.

Following the exploration of the problem statement and the underlying motivations in Chapter 1, the subsequent chapter, Chapter 2, offers a theoretical analysis. This analysis demonstrates that increasing delay on shorter paths has the potential to yield further improvements in performance, boosting the benefits achieved only using *clock skew* and *retiming*.

Chapter 3 presents both the original and the updated versions of this methodology. The original one takes as input the timing data from the post-layout netlist of the FF-based design. The updated one presents a variation of the original *Mix & Latch* flow that aims to reduce runtime and increase performance to catch up with conventional retiming methods. First, we introduced tolerances in the timing analysis to give *Mix & Latch* more freedom to choose the NTL positions. The PTL netlist layout was then avoided to reduce execution time by performing the timing analysis required by *Mix & Latch* to determine the position of NTLs in the netlist earlier, in the post-synthesis phase.

Chapter 4 presents the experimental results collected for the original and the updated versions of the methodology. The testing of the original flow shows *simultaneously smaller area and higher working frequency* for all the circuits that we considered, except for `s38417`, `s9234`, `s1196`, and `s13207`, where *only performance*

is significantly improved. For most cryptography circuits, the area improvement exceeds 1.3 X.

Even though our approach does not aim at improving area and uses just one clock phase, our area reduction is in some cases comparable to that of a recent work [14], which uses three clock phases,rather than just one, and focuses on area optimization.

As an example benchmark, we tested the updated *Mix & Latch* flow on a *Zeroriscy* RISC-V core. The results show a 25 % clock frequency improvement over the original flow and a 7.5 % over retiming, with an average 5.53 % lower power consumption and 12.63 % lower area occupation. The PPA improvements prove that the *Mix & Latch* methodology can challenge traditional retiming-based techniques while avoiding the drawbacks that make the latter inapplicable to industrial design flows, i.e., by allowing combinational equivalence checking to be used throughout the flow.

Future work will focus on further improving parameter selection (e.g., depending on circuit topology), on extending the evaluation to other circuits, and on analyzing the applicable DFT methodologies.

References

[1] Vojin G. Oklobdzija, editor. *Digital Design and Fabrication*. CRC Press, December 2017.

[2] Shi-Zheng Eric Lin, Chieh Changfan, Yu-Chin Hsu, and Fur-Shing Tsai. Optimal time borrowing analysis and timing budgeting optimization for latch-based designs. *ACM Trans. Des. Autom. Electron. Syst.*, 7(1):217–230, jan 2002.

[3] E.G. Friedman. Clock distribution networks in synchronous digital integrated circuits. *Proceedings of the IEEE*, 89(5):665–692, 2001.

[4] Matthew Fojtik, David Fick, Yejoong Kim, Nathaniel Pinckney, David Money Harris, David Blaauw, and Dennis Sylvester. Bubble razor: Eliminating timing margins in an arm cortex-m3 processor in 45 nm cmos using architecturally independent error detection and correction. *IEEE Journal of Solid-State Circuits*, 48(1):66–81, 2013.

[5] Mihir R. Choudhury, Vikas Chandra, Robert C. Aitken, and Kartik Mohanram. Time-borrowing circuit designs and hardware prototyping for timing error resilience. *IEEE Transactions on Computers*, 63(2):497–509, 2014.

[6] Kamlesh Singh, Hailong Jiao, Jos Huisken, Hamed Fatemi, and José Pineda de Gyvez. Low power latch based design with smart retiming. In *2018 19th International Symposium on Quality Electronic Design (ISQED)*, pages 329–334, 2018.

[7] Nik Azman Nik Hassan, Asrulnizam Bin Abd Manaf, and Leong Chan Ming. Optimization of circuitry for power and area efficiency by using combination between latch and register. In *2011 IEEE International Conference on Computer Applications and Industrial Electronics (ICCAIE)*, pages 240–244, 2011.

[8] Marc Pons, Thanh-Chau Le, Claude Arm, Daniel Séverac, Jean-Luc Nagel, Marc Morgan, and Stéphane Emery. Sub-threshold latch-based icyflex2 32-bit processor with wide supply range operation. In *2016 46th European Solid-State Device Research Conference (ESSDERC)*, pages 33–36, 2016.

[9] Aaron P. Hurst and Robert K. Brayton. The advantages of latch-based design under process variation. In *Proceedings of the IWLS*, 2006.

[10] B. Taskin and I.S. Kourtev. Time borrowing and clock skew scheduling effects on multi-phase level-sensitive circuits. In *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, volume 2, pages II–617, 2004.

[11] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. Optimal Clocking of Synchronous Systems. In *TAU90—ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, University of British Columbia, Vancouver, August 1990.

[12] K.A. Sakallah, T.N. Mudge, and O.A. Olukotun. Analysis and design of latch-controlled synchronous digital circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(3):322–333, 1992.

[13] Yanqing Zhang and Benton H. Calhoun. Hold time closure for subthreshold circuits using a two-phase, latch based timing method. In *2013 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, pages 1–2, 2013.

[14] Huimei Cheng, Xi Li, Yichen Gu, and Peter A. Beerel. Converting flip-flop to clock-gated 3-phase latch-based designs using graph-based retiming. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1, 2021.

[15] Youngsoo Shin and Seungwhun Paik. Pulsed-latch circuits: A new dimension in asic design. *IEEE Design Test of Computers*, 28(6):50–57, 2011.

[16] Jin-Fa Lin. Low-power pulse-triggered flip-flop design based on a signal feed-through. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(1):181–185, 2014.

[17] Hyein Lee, Seungwhun Paik, and Youngsoo Shin. Pulse width allocation and clock skew scheduling: Optimizing sequential circuits based on pulsed latches. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(3):355–366, 2010.

[18] Seungwhun Paik, Lee-eun Yu, and Youngsoo Shin. Statistical time borrowing for pulsed-latch circuit designs. In *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 675–680, 2010.

[19] Thomas Baumann, Doris Schmitt-Landsiedel, and Christian Pacha. Architectural assessment of design techniques to improve speed and robustness in embedded microprocessors. In *2009 46th ACM/IEEE Design Automation Conference*, pages 947–950, 2009.

[20] Seonggwan Lee, Seungwhun Paik, and Youngsoo Shin. Retiming and time borrowing: Optimizing high-performance pulsed-latch-based circuits. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, ICCAD '09, page 375–380, New York, NY, USA, 2009. Association for Computing Machinery.

[21] Seungwhun Paik, Lee-eun Yu, and Youngsoo Shin. Statistical time borrowing for pulsed-latch circuit designs. In *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 675–680, 2010.

[22] B. Taskin and I.S. Kourtev. Time borrowing and clock skew scheduling effects on multi-phase level-sensitive circuits. In *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, volume 2, pages II–617, 2004.

[23] K. Yoshikawa, K. Kanamaru, S. Inui, Y. Hagihara, Y. Nakamura, and T. Yoshimura. Timing optimization by replacing flip-flops to latches. In *ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No.04EX753)*, pages 186–191, 2004.

[24] Satyamurthy Pullela, Noel Menezes, and Lawrence T. Pillage. Reliable non-zero skew clock trees using wire width optimization. In *Proceedings of the 30th International Design Automation Conference*, DAC '93, page 165–170, New York, NY, USA, 1993. Association for Computing Machinery.

[25] J.P. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, 39(7):945–951, 1990.

[26] J.L. Neves and E.G. Friedman. Design methodology for synthesizing clock distribution networks exploiting nonzero localized clock skew. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 4(2):286–291, 1996.

[27] R.B. Deokar and S.S. Sapatnekar. A graph-theoretic approach to clock skew optimization. In *1994 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 1, pages 407–410 vol.1, 1994.

[28] D.A. Joy and Maciej Ciesielski. Clock period minimization with wave pipelining. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Apr 1993.

[29] S.S. Sapatnekar and R.B. Deokar. Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(10):1237–1248, 1996.

[30] Narendra V. Shenoy, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Minimum padding to satisfy short path constraints. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '93, page 156–161, Washington, DC, USA, 1993. IEEE Computer Society Press.

[31] Cunxi Yu, Chau-Chin Huang, Gi-Joon Nam, Mihir Choudhury, Victor N. Kravets, Andrew Sullivan, Maciej Ciesielski, and Giovanni De Micheli. End-to-end industrial study of retiming. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 203–208, 2018.

[32] Kun Young Chung and Sandeep K. Gupta. Design and test of latch-based circuits to maximize performance, yield, and delay test quality. In *2010 IEEE International Test Conference*, pages 1–10, 2010.

[33] Jie-Hong R. Jiang and Robert K. Brayton. Retiming and resynthesis: A complexity perspective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(12):2674–2686, 2006.

[34] Charles E Leiserson and James B Saxe. Retiming synchronous circuitry. In *Algorithmica*, volume 6, page 5–35, 1988.

[35] K. Yoshikawa, K. Kanamaru, S. Inui, Y. Hagihara, Y. Nakamura, and T. Yoshimura. Timing optimization by replacing flip-flops to latches. In *ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No.04EX753)*, pages 186–191, 2004.

[36] Filippo Minnella, Jordi Cortadella, Mario R. Casu, Mihai T. Lazarescu, and Luciano Lavagno. Mix & latch: An optimization flow for high-performance designs with single-clock mixed-polarity latches and flip-flops. *IEEE Access*, 11:35830–35840, 2023.

[37] Optimization with pulp. https://coin-or.github.io/pulp/, 2009.

[38] MIT-LL. Common evaluation platform (cep). https://github.com/mit-ll/CEP.git, 2021.

[39] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Circuits and Systems, 1989., IEEE International Symposium on*, pages 1929–1934 vol.3, May 1989.

[40] F. Corno, M.S. Reorda, and G. Squillero. Rt-level itc'99 benchmarks and first atpg results. *Design Test of Computers, IEEE*, 17(3):44–53, Jul 2000.

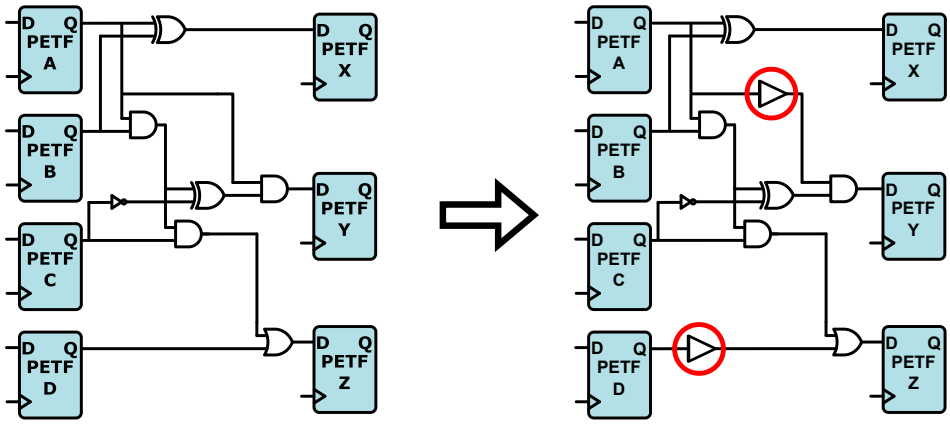
[41] Pasquale Davide Schiavone, Francesco Conti, Davide Rossi, Michael Gautschi, Antonio Pullini, Eric Flamand, and Luca Benini. Slow and steady wins the race? a comparison of ultra-low-power risc-v cores for internet-of-things applications. In *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 1–8, 2017.

Appendix A

Formal verification of the *Mix & Latch* methodology

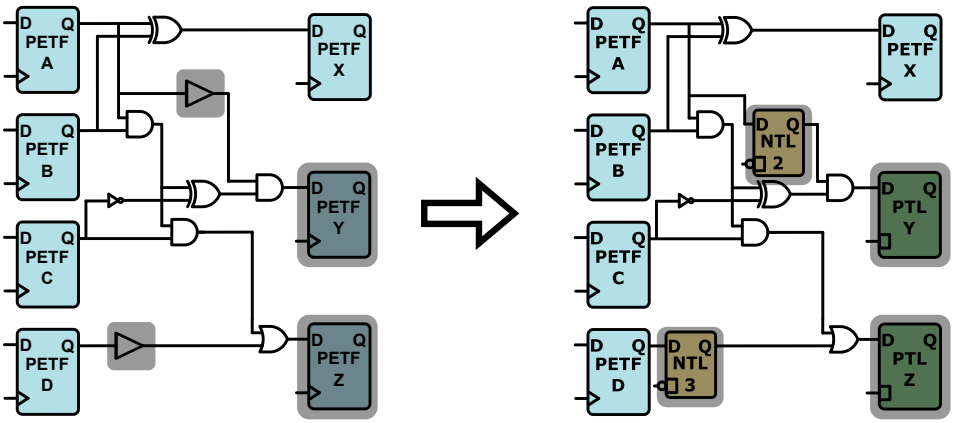
The implementation flow checks for system behavior through formal verification. It is implemented with a two-steps checking procedure:

- **Adding buffers**: As shown in Fig. A.1a, the first step is to modify the PETFs based post-synthesis netlist adding buffers in the same position computed for the NTLs. The original PETFs-based netlist is verified against the one with the additional buffers.



(a) Adding buffers

- **Black box declaration**: As shown in Fig. A.2a, the second step is to declare as black-boxes:



(a) Black box declaration

- PETFs-based netlist: the buffers and the modified PETFs;
- *Mix & Latch* netlist: the NTLs and the elements transformed in PTLs or NETFs;

The PETFs-based netlist with the inserted buffers is now verified against the *Mix & Latch* netlist.

The example shows the procedure on a *Worst-Path* flow example.