POLITECNICO DI TORINO

Master's Degree in Electronic Engineering

Master's Degree Thesis

Exploring the Razor Approach for Better Than Worst-Case Design in Latency-Insensitive Digital Circuits

Supervisors
Prof. Luciano Lavagno
Phd. Filippo Minnella

Candidate
Marco Massetti

December 2023

Abstract

The advancement of technology has led to an exponential growth in the demand for computing power, pushing to the limit the capabilities of existing hardware. Digital circuits have met these performance requirements by architectural improvements and by technology scaling. But, as semiconductor technology continues to shrink, the sensitivity of the circuits to small deviations in manufacturing processes, supply voltage, and operating temperature becomes more pronounced.
Traditional design methodologies that rely on worst-case scenarios may result in sub-optimal performance or excessive power consumption due to their overly conservative nature.
Instead of always considering the worst-case, it is possible to design considering a better-than-worst-case condition, to allow the generation of errors, and implement a method to correct them. If the correction of errors does not introduce a too large overhead, a meaningful increase in efficiency is achievable.
The objective of this thesis work is to apply the Razor error detection and correction technique, tackling the timing violations caused by the introduction of the error detection by utilizing a novel optimization flow called Mix & Latch. To reduce the impact on performance caused by the error correction, Razor is applied to latency-insensitive designs, in such designs an error in a block of logic will slow down only that block, while the rest of the circuit will operate without interruptions. Given that latency-insensitive designs are commonly created with a high-level synthesis framework, this paradigm is exploited to create the designs used in this thesis.
The results of the flow are validated using post-layout simulations to evaluate the area, performance and power overhead caused by the addition of error detection and correction to the designs. Both the possible gains in performance attainable by increasing the clock frequency and the reduction in power consumption achievable by voltage scaling are studied.

Table of Contents

List of Figures

List of Tables

List of Acronyms

PVT Process Voltage Temperature

TRC Tunable Replica Circuit

EDA Electronic Design Automation

PETF Positive clock Edge Triggered Flip-flop

NETF Negative clock Edge Triggered Flip-flop

PTL Positive clock phase Transparent Latch

NTL Negative clock phase Transparent Latch

STA Static Timing Analysis

ILP Integer Linear Programming

LI Latency Insensitive

SLD System Level Design

HLS High Level Synthesis

RTL Register Transfer Level

HDL Hardware Description Language

RCA Ripple Carry Adder

FIR Finite Impulse Response

FIFO First In First Out

Chapter 1

Introduction

1.1 Overview

The relentless advancement of technology in recent years has led to an exponential growth in the demand for computing power across various sectors. From artificial intelligence to data analytics and complex simulations, the computational requirements have consistently pushed to the limit the capabilities of existing hardware. In the last years digital circuits have met these performance requirements by architectural improvements and by technology scaling, but as semiconductor technology continues to shrink and more transistors are integrated on a single chip, power dissipation has emerged as a critical bottleneck [1].
Moreover, as feature sizes continue to shrink, the sensitivity of the circuit to small deviations in manufacturing processes, supply voltage, and operating temperature (PVT) becomes more pronounced [2]. This poses a formidable challenge for circuit design, as the traditional design methodologies that rely on worst-case scenarios may result in suboptimal performance or excessive power consumption due to its overly conservative nature.

1.2 Alternatives to the worst-case approach

Since the performance of every transistor can change greatly even on the same die [3], in the worst-case approach the design needs to be realized adding safety margins to consider these variations, in this way there is the certainty that the device will work, but, since in most of cases not all the transistors will have the worst possible characteristics, it also causes excessive requirements.
Adaptive design techniques have emerged as a response to the challenges posed by increasing PVT variations and the limitations of traditional worst-case design

methodologies. Since the variability of the process impacts the performances in a static way, in the sense that it does not change during the lifetime of the device, it can be compensated using adaptive design techniques; this approach consists in the variation of the working voltage and frequency at run-time, this way it is possible to adjust the operating point depending on the performance of the transistors. One of these techniques is the use of tunable replica circuits (TRCs) [4] also called "canary circuits", here a replica of the critical path is used to check how much it is possible to increase the clock frequency (or reduce the supply voltage) before the circuit fails. The downside of this approach is that the replica path must always fail before the real critical path to guarantee the absence of errors, so also with adaptive designs some safety margins must be included. Moreover, this technique can compensate only for some of the issues to which digital circuits are subject. Supply voltage fluctuation, temperature variations, IR drop, clock jitter, crosstalk noise; they are particularly troublesome because they can change quickly at run-time and be localized for a specific region of the die, so they cannot be evaluated by techniques like the canary circuits.
A solution to these problems is using a better than worst-case design approach. In this case the generation of errors is allowed and a method to correct them is implemented. This technique is already used for communications to deal with data corruption during transmission across noisy channels, but in these applications usually algorithmic solutions, such as error correcting codes are used; but for general purpose computing this solution is likely to add an excessive area and power overhead, for this type of designs it is possible to use solutions like Razor.

1.3 Razor

Razor [5] is a timing speculation technique that can be used to dynamically adjust the working conditions of digital circuits according to the number of errors that it detects. The idea at the base of this design methodology is that at the end of the setup critical paths it is possible to sample the data at two different times, it is sampled a first time by the flip-flop originally present in the design and it is sampled a second time by a "shadow latch", these two sequential elements share the same

data signal but they use different clocks. To be able to detect errors caused by late arrival of the data the two instants where the input is sampled are shifted in time one from the other, the flip-flop will sample speculating that the data has already concluded the propagation along the preceding combinational logic stage, while the shadow latch will sample the data later, at a time that ensures that the correct data has reached the end of the combinational path. The output of the shadow latch is not directly connected to the rest of the circuit, but it is compared to the output of the main flip-flop, if there is a difference in the two signals it means that the timing speculation was too aggressive, and that the error must be corrected. An example of error detection cell and of its behavior are depicted in figure 1.1.

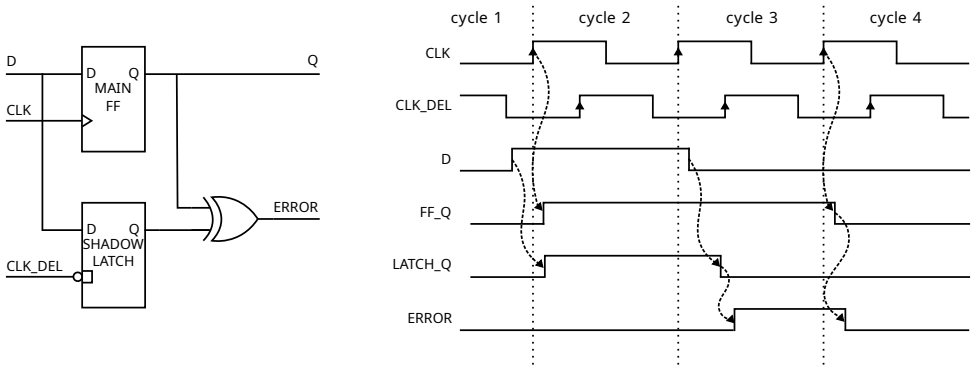

Figure 1.1: Behavior of a Razor error detection cell

Using this technique it is possible to increase the working frequency (or decrease the supply voltage) until errors are detected, moreover, if the losses in performance caused by the recovery mechanism are lower than the gains caused by the increase in frequency, it is possible to accept the errors until the error recovery overhead becomes too prominent.

It is clear how this solution is advantageous compared to static adaptive design techniques, it allows to eliminate completely the safety margins and, since it is applied directly on the critical paths, it can also detect errors caused by fast changes and it is not dependent on its location in the die.

However, some considerations must be done while applying this technique: with the increase in frequency there are problems of metastability on the flip-flops; and since the clock for the shadow latch is delayed there is the risk of hold violations. If the

clock frequency is higher than the nominal one there is the possibility that the data at the input of the flip-flop will change near the sampling edge of the clock, this will cause the output to become metastable, if no countermeasures are implemented this metastability will propagate in the following stages of logic and in the error signal. The metastability problem cannot be completely resolved, but its probability to propagate can be reduced using skewed gates and double sampling the flip-flop output signal. For the hold violations the situation is more complicated, the clock provided to the shadow latch must be delayed enough to be able to sample the correct data also with a higher working frequency, this means that if a higher frequency increase is desired, the clock to the latch must be delayed further. The problem with a higher clock delay is that the data propagating along short paths that ends in shadow latches must not reach the latch before the data launched in the previous clock cycle has been correctly sampled. While both the issues must be tackled to obtain a working design, this thesis will focus on the hold constraint issue.

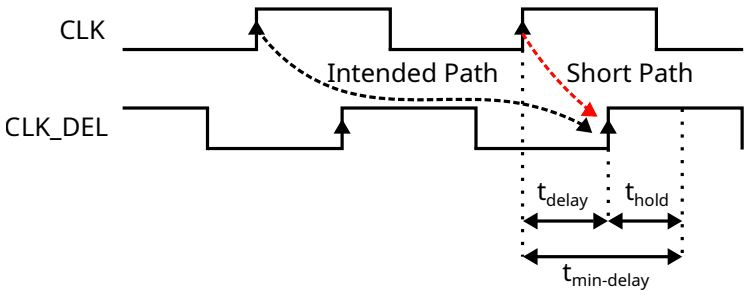

Figure 1.2: Hold timing constraint

To resolve the hold violations it is required to increase the delay of the shortest paths, this can be achieved inserting buffers, however this solution comes at the cost of an increased area and power overhead. A second possible solution to the problem is inserting latches transparent on the clock level opposite to the shadow latches on the short paths, this solution has a lower impact on the system performance, but it cannot be automatically done by commercial electronic design automation (EDA) software. Automated flows that can be used to place latches to correct hold violations do exist, the one that is exploited in this thesis is the Mix & Latch flow.

1.4 Mix & Latch

Mix & Latch [6] is an optimization flow that exploits mixed polarity latches and flip-flops to increase the working frequency of digital circuits. Sequential circuits can store data using two different elements: flop-flops and latches, while the later offers reduced area, power and delay than the former, it also has more complex timing constraints and this limits theirs use in commercial applications. Mix & Latch is able to automatically convert a flip-flop based design to a latch based one, that is faster and can also have a reduced area compared to the original one. Traditional latch designs require particular clocking schemes to avoid the hold violations caused by the transparent phase of the latches: it is possible to reduce the clock duty cycle to obtain short pulses, this solves the hold issues, but it also reduces the performance benefits of time borrowing; a second choice is to use two non-overlapped clocks and alternate the one used from the paths between latches, but this causes the design procedure to become difficult. Mix & Latch solves the hold timing violations by using a single clock with 50% duty cycle and inserting latch transparent on the negative clock level along the hold critical paths.
The flow is represented in figure 1.3 and proceeds according to these steps:

- From the original positive edge triggered flip-flop (PETF) based netlist all the sequential elements are replaced with latches transparent on the positive clock phase (PTL), this will cause the generation of a netlist with hold violations due to the paths between the PTLs.
- Using static timing analysis (STA) a graph representation of the timing of the entire circuit is created, a sub-graph with only the pins and edges that belongs to paths violating the hold constraint is also created.
- Using an integer linear programming (ILP) algorithm [7] the locations where latches transparent on the negative clock phase (NTL) must be inserted to fix the hold violations are found. Solving an ILP generally requires a high runtime, but in this particular case it is structured in a way close to a max-flow min-cut problem, which has polynomial complexity.
- In the points where there are adjacent PTLs and NTLs, the two sequential elements are merged in flip-flops sensible to the rising edge or falling edge to

reduce the area occupation.

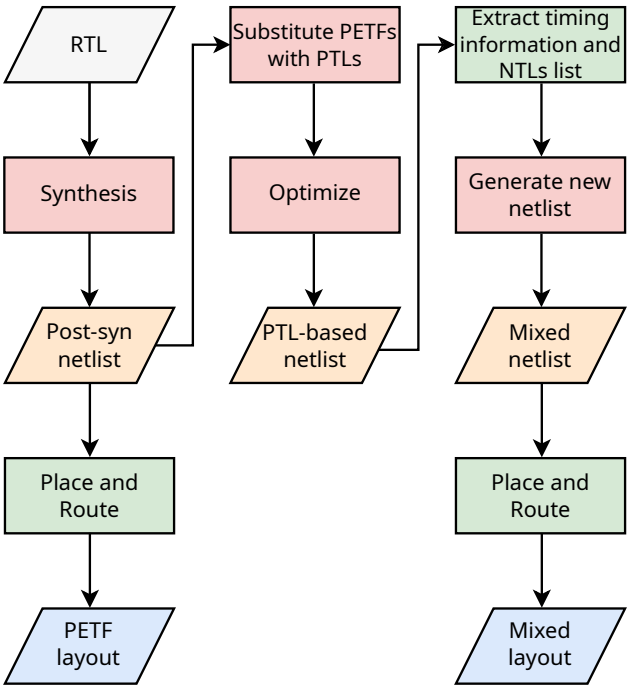

Figure 1.3: Mix & Latch flow

Experimental results show that after the flow the circuits have simultaneously smaller area and higher working frequency compared to their original form.
In this thesis this flow is not exploited for these improvements, but it is useful due to its capability of fixing the hold violations via the introduction of latches along the hold critical paths. This solution is advantageous compared to the traditional buffer insertion due to its reduced area and power overhead.

1.5 Latency-insensitive design

The Latency-insensitive (LI) design approach is used in digital systems to minimize the impact of timing delays and ensure reliable operation even in the presence of variable latency. This type of circuit has been introduced as solution for the challenging task of guarantee consistent timing in complex structures such as system-on-chip, where the interconnections represent a significant cause of delay [8], but it can also

be used for the design of traditional pipelined logic.
LI circuits are designed in such a way that they operate correctly and efficiently regardless of the time it takes for signals to propagate through various parts of the circuit, this means that they don't depend on precise timing, making them resilient to variable delays. This is achieved using communication protocols between different blocks of the circuit, they involve the exchanging of signals between sender and receiver to confirm data transfer. Such channels can space from simple wires to FIFOs that can manage the different data production/request of blocks and allow for different clock domains in the design. Each block of the circuit can operate independently, and these protocols ensure that a block that finishes its operation doesn't proceed in the next operation until the receiver is ready to accept the data that has been just produced.

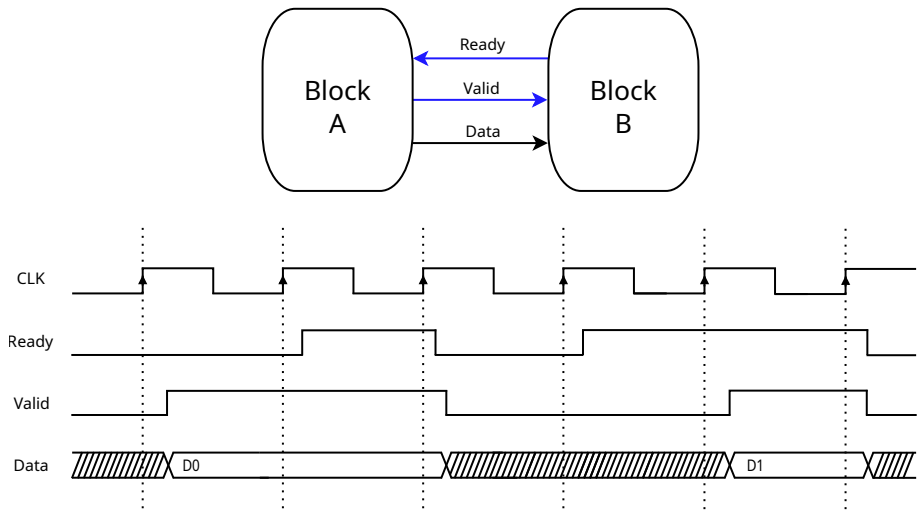

Figure 1.4: Communication between blocks in the design

In systems using Razor, maintaining correct operation and avoiding degrading the performance when errors are detected is crucial. Applying Razor to LI designs avoids such issues, the stalling of instruction execution that is caused by the error correction can be handled by the flow control mechanism inherent in the operation of the communication channels. This avoids the need for a global stall signal that in large systems can impact the maximum operating frequency. Moreover, since the stalls are localized in a single block rather than to the entire system, the loss of performance

is limited and, if other blocks require more clock cycles to terminate their operation than the one that has committed the error, the stall does not cause any performance degradation. Finally, another advantage of leveraging the LI methodology is that, if a block of the design is a bottleneck for the performance due to its low throughput, it is possible to speedup only that block by providing it with a faster clock and by inserting Razor cells. All of this can be done without any impact on the rest of the design by using clock domain crossing FIFOs at the interfaces of that block.
A common use case for the LI approach is in designs created with flows that exploit high-level synthesis (HLS), in such design flows the HLS tool automatically schedules the operations, and this means that the latency of each logic block is not known a priori.

1.6 High-level synthesis

The increase of complexity of modern devices has led to the exploration of new design methodologies to accelerate the design procedure and to reduce the time to market. System-level design (SLD) is a valid alternative to the traditional bottom-up approach, this design paradigm involves creating algorithmic descriptions using high-level programming languages (C, C++, ...) and then, using HLS tools, converting these descriptions into a hardware register transfer level (RTL) representation.
In addition to speed up design, this methodology also enables to reduce the time required for verification. It is now possible to execute the software that describes the device instead of running a slower hardware description language (HDL) simulation. Moreover, design space exploration is also easier, from the same algorithmic description it is possible to obtain different hardware realizations by specifying to the HLS tool the desired characteristics (latency, throughput, area, ...).
Because SLD is often used to create LI designs, and it is always becoming more relevant due to its use for machine learning and digital signal processing application, this design methodology is used to produce test circuits to which apply the flow that has been developed.

Chapter 2

Related Work

Several methods that focus on the reduction of the timing margin have been previously studied. The TRC technique [9, 4] can reduce, but not completely eliminate, the guard bands, and it is not capable of handling fast changing and local timing variations. The use of Razor technique [5, 10, 11] allows to eliminate completely the timing margin and it provides *in situ* error detection, therefore also local timing variations can be handled. The error detection circuits proposed in [5] and [11] require two clock signals to function, the error detection architecture that is exploited in this thesis is the "DSTB" proposed in [12] that can work with only one clock.
Regarding the correction of the hold violations, the most used solution is to insert chain of buffers to increase the delay of short paths, other solutions involve the use of latches to reduce or eliminate the padding required to the short paths. Using latches the solutions that have been studied vary from using only one clock [6], to others using two [13, 11] or three [14] clock phases; with the addition of more clock phases the timing closure becomes easier, but this causes an increase of complexity in the clock tree and of its power consumption.
To correct the errors that are detected different methods have been used: the simplest is global clock gating [5], but it may not be usable in larger circuits; solutions like counterflow pipelining [5] and instruction replay [12] can be scaled for larger systems but they add large area and power overhead. The idea of localizing the error recovery to smaller circuit's blocks has been studied in [15] and [16], where Razor is applied respectively to SoCs and LI designs using communication protocols, and stall mechanisms, different from the ones exploited in this thesis.

Chapter 3

Implementation

The complete flow capable of automatically insert Razor error detection to a circuit, solve the hold violations, and implement an error recovery mechanism, has been realized in two development steps:

1. Error detection: development of a flow capable of inserting the Razor cells at the end of the setup critical paths, fix the hold violations and adding a signal that indicates if an error has occurred.
2. Error detection and correction: expanding the flow by developing a mechanism to correct the errors detected by Razor.

3.1 Error detection

For this first step the design flow has been realized in order to be able, starting from a valid RTL description of the hardware, to produce a working gate level netlist where the flip-flops at the end of the setup critical paths have been replaced with Razor cells, and where the error signals of all the Razor cells are merged together and connected to an output port. At this stage no error correction is planned.

3.1.1 Razor architecture

The type of Razor circuit that has been used and an example of its behavior, are shown in figure 3.1. Compared to the error detection circuit displayed before (figure 1.1), in this structure only one clock signal is used for both the sequential elements, the output data is taken from a PTL instead of a flip-flop, and the shadow latch has been replaced by a PETF.

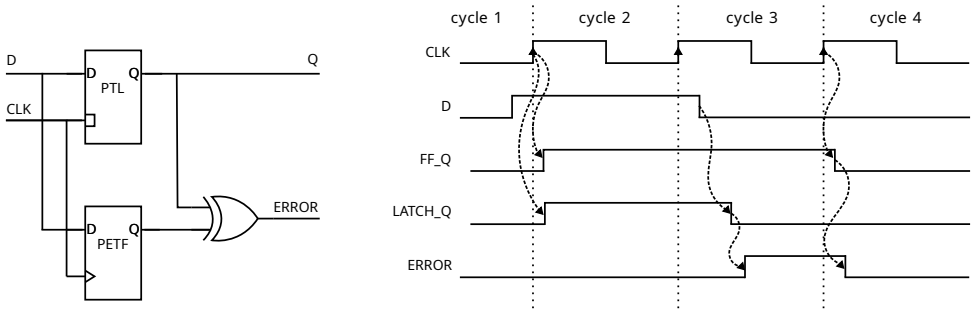

Figure 3.1: Error detection circuit

Also in this case the element that speculates over the arrival time of the data is the flip-flop, it samples the incoming data on the rising clock edge, meanwhile, the PTL remains transparent and permits the flow of data to its output for the entire high clock phase. When the transparency phase of the latch has ended, it is possible to detect if data has arrived late by comparing the outputs of the two sequential elements.

Compared to the original Razor cell there is a lower area and power overhead since only one clock signal is required. Moreover, the main advantage of this type of circuit is that, since the output of the latch is the input for the next stage of logic, the possibility of having a metastable signal on the datapath is eliminated. It is still possible to have a metastable error signal since the sampling for the flip-flop occurs earlier and the data can change in proximity of the clock edge. Another advantage of this architecture is that the output data has always the correct value because the timing speculation is done on the FF, this will be important for the error correction phase because it makes possible to ensure the correct operation of the circuit by issuing only a one clock cycle stall.

To ensure the correct behavior of this circuit it is mandatory to guarantee a maximum and minimum arrival time for the data to the Razor cell.

While using this structure the setup timing constrains for the EDA tolls are still based on the rising clock edge, and the window for which it is possible to detect errors is equal to the high clock phase. The maximum path delay in the worst conditions (slow PVT corner, clock jitter, ...) that allows for the correct detection of error, must guarantee that the maximum data arrival time to the endpoint will remain inside the transparency phase of the latch, this delay can be calculated with the equation 3.1.

$$T_{max} \leq T_{cycle} \cdot (1 + DC) - T_{setup,ptl} \tag{3.1}$$

Where T_{cycle} is the clock period, DC is the duty cycle, $T_{setup,ptl}$ is the setup time of the latch.
To avoid that data propagating along short paths will be sampled by the latch during the wrong clock cycle, the minimum delay of all the paths the end up in a Razor cell must be at least equal to the transparent phase of the latch, as defined in the equation 3.2.

$$T_{min} \geq T_{cycle} \cdot DC + T_{hold,ptl} \tag{3.2}$$

Where $T_{hold,ptl}$ is the hold time of the latch based on the falling clock edge.
A visual representation of these two timing constraints is depicted in figure 3.2.

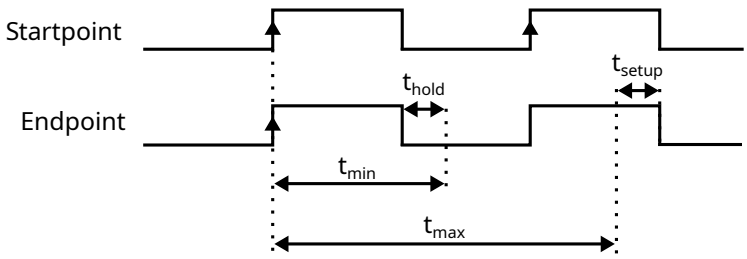

Figure 3.2: Timing constraints

3.1.2 Edits to the Mix & Latch flow

The flow that has been implemented for Mix & Latch is a suitable starting point for the flow required for the insertion of Razor to the circuit, however, some modifications have been required. The modified flow can be represented with the flowchart in figure 3.3, the edits made from the original flow are explained below.

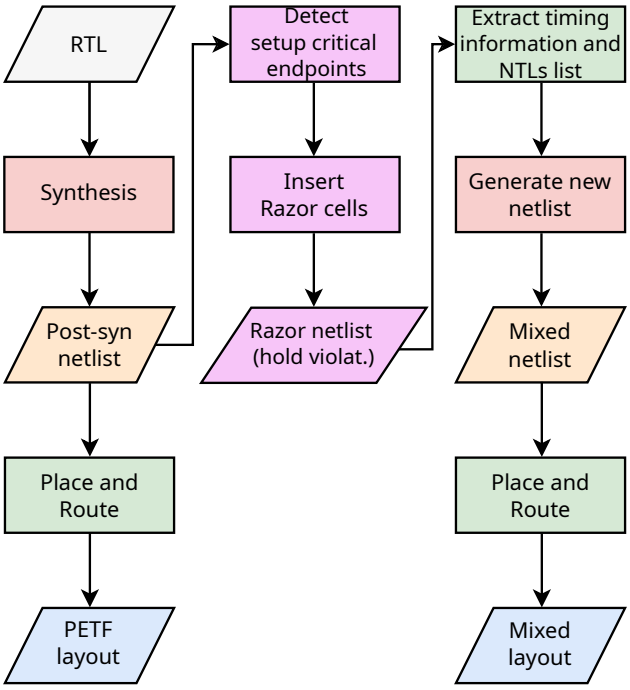

Figure 3.3: Modified Mix & Latch flow

Insertion of Razor endpoints to setup critical paths

In the original flow, after the first synthesis, all the flip-flops are replaced with latches. This step has been removed, instead, all the endpoints of combinational paths that have a setup slack less than a threshold specifiable by the user are detected. These are the flip-flops that are more prone to capture a wrong value in case of slow propagation of the signals in the datapath, for this reason these flip-flops will be replaced with Razor cells.
To enable the error detection, the netlist is modified replacing the critical FF with Razor cells, then, if more than one FF has been replaced, all the error signals of the Razor cells are merged together using a tree of OR gates. Finally, an output port is added to the design, and the global error signal is connected to it. This replacing step is summarized in figure 3.4.

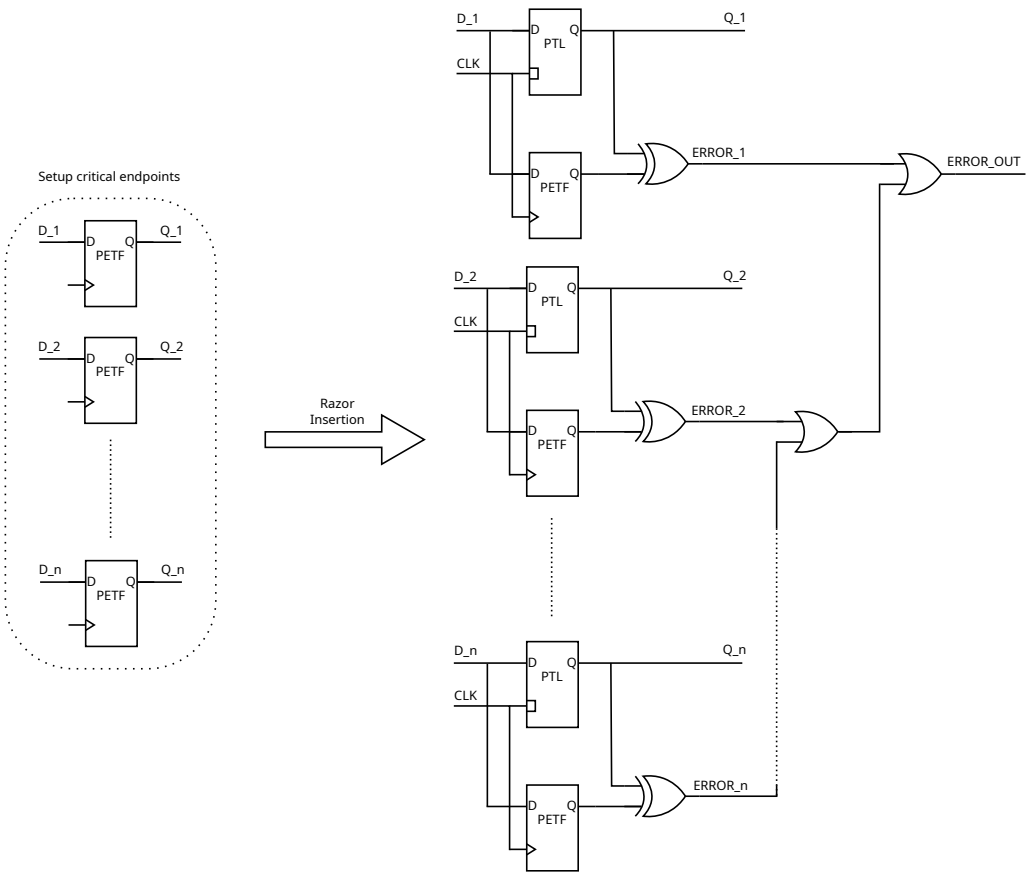

Figure 3.4: Insertion of the Razor cells

Additional timing constraints

After the insertion of Razor, to guarantee that the final device will behave as expected, it is required to add a few timing constraints for the EDA tools:

- Time borrowing for the PTL of Razor cells: since the time available for each combinatorial path must not change from the original design, it is required to avoid the time borrowing at the PTL added to the circuit, this way the setup constraint is still equal to the rising clock edge
- Time borrowing for NTL: by default the Mix & Latch flow considers that a time borrowing equal to the entire low clock phase is possible on the NTLs that it will add to resolve the hold violations. However, this can lead to the condition were the NTLs become the endpoints for the setup critical paths.

In this condition if there is a slow propagation of the data, it is possible that the transparency window of the NTL closes before the data has been able to propagate through the latch, as in figure 3.5. In such cases the late arriving of the data will not be detectable by the Razor cell at the end of the path.

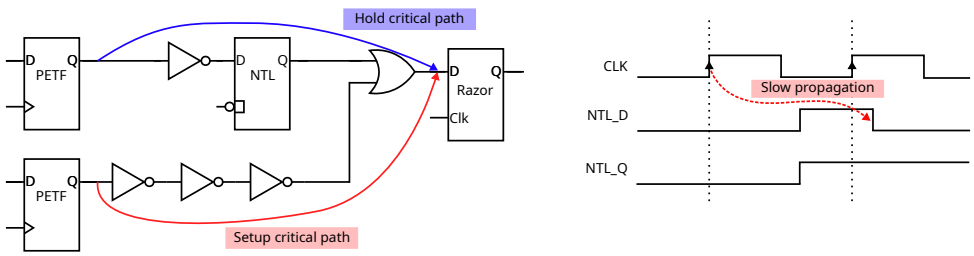

Figure 3.5: Error caused by NTL sampling

To prevent this condition, the time borrowing margin for in NTLs is reduced to a value specified by the user, this will force the Mix & Latch flow to reduce the length of the path ending in the NTLs. In case of slow data propagation there is a still a guard band time in which errors are avoided. Since the latches are placed only on short paths this reduction of time does not introduce a reduction in the design's performance. The effect of this constraint is displayed in figure 3.6

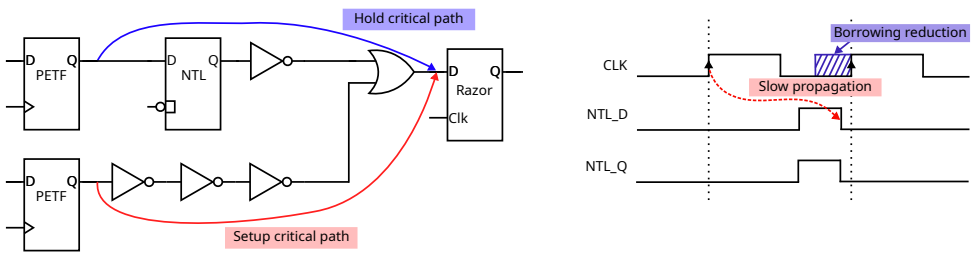

Figure 3.6: Effect of reduced NTL time borrowing

- Minimum setup slack for non-Razor endpoints: as explained before, the Razor technique works only if the Razor elements are placed at the end of the setup critical paths. Since the design flow is composed by multiple synthesis and a final layout, it is possible to obtain different circuits that satisfy the design constraints, this can be a problem since paths different from the ones that ends

in Razor cells can become setup critical. Since after the first synthesis all the paths that have not been selected to become sampled by Razor have a setup slack higher that the threshold chosen by the designer, a constraint is added for the following steps to force all the non-Razor paths to have a setup slack at least equal to that threshold, this constraint can be formulated as in equation 3.3.

$$T_{max,non-razor} \leq T_{cycle} \cdot (1 - Threshold) \tag{3.3}$$

For instance, if all the endpoints that have a setup slack less than 30% of clock period are selected to be replaced with Razor cells, than all the other endpoints must always have a setup slack greater than 30% of the clock period

Extraction of timing information

Since the design has different timing constraints from what expected from the original Mix & Latch flow, the generation of the timing graph requires some changes to match the newly added timing constraint. The original flow replaced all the sequential elements of the design with PTLs, and it considered the possibility of a maximum time borrowing equal to the high clock phase. For the reasons already discussed, after the insertion of Razor it is no more possible to have time borrowing, therefore the maximum time borrowing is zeroed. A second difference is that the time borrowing window available for the NTLs is no more equal to the full low phase of the clock, so it is now considered equal to the value specified by the user.

Avoiding merging of NTL with Razor PTL

At the end of the Mix & Latch flow, merging of adjacent PTLs and NTLs is done to reduce the circuit's area. In this step if a PTL is connected to the output of a NTL a PETF is placed instead of the two latches, while if an NTL is connected to the output of a PTL a negative edge triggered flop (NETF) is inserted.
This step must now be avoided for the latches that form the Razor cells, the effectiveness of the error detection is based on the fact that each Razor cell is formed by a PETF and a PTL, if this structure is altered in any way the error detection mechanism will no longer work.

3.2 Error detection and correction

The goal for this second development step is to expand the flow previously described to be able to correct the errors that it detects.
The error correction has been realized in order to work with latency-insensitive circuits. This type of design is composed by multiple logic blocks that can work concurrently on different sets of data, the data transfers between different blocks is managed by synchronous handshakes. This type of structure is often used in designs generated with HLS tools, in this thesis the designs has been realized using the Mentor Catapult HLS tool.
In the type of Razor cell that has been chosen for this thesis the value of the output data signal is always correct (because the timing speculation is done by the FF on the error path). However, a late arrival of the data can cause the following stage of logic to not have enough time to conclude correctly its operation, for this reason, it is required to stall all the registers of the block where the error has been detected for one clock cycle, like shown in figure 3.7. Since the circuit works in dataflow style with LI interfaces between the blocks, it is also required to block the handshake signals to avoid that other blocks in the design can read or send data to the stalled block.

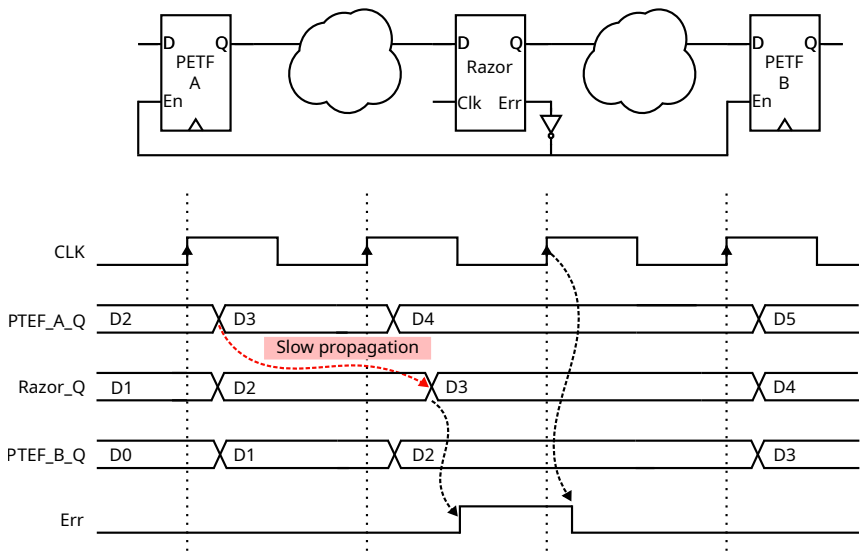

Figure 3.7: Behavior in case of error

3.2.1 Razor architecture

The basic error detection cell is still the same as the previous circuit, but, in order to correct the error, more considerations have to be made. The complete error path is depicted in figure 3.8, each section of the circuit is explained in detail below.

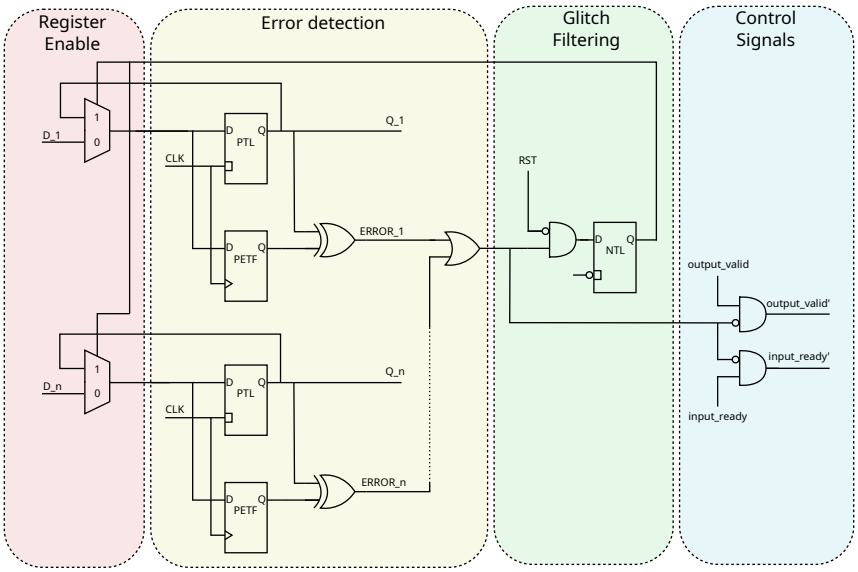

Figure 3.8: Error path for dataflow circuits

Register enable

To stall the registers of the block that has committed a timing violation, all the flip-flops in that block must have an enable signal to select if it is required to sample a new input or to maintain the last value previously stored. While for all the non-Razor registers in the design this can be achieved by using a traditional FF with enable port and connecting it to the error signal, the Razor cells must use a different approach. For razor cells the enable functionality is achieved by adding a 2-to-1 multiplexer to select which signal has to be sampled the next clock edge, one input is connected to the input data line of the Razor cell and the second input is a feedback from the output of the Razor PTL. As for the other FF, the enable function is managed by the error signal of the block. This type of structure can simultaneously stall the pipeline and restart it after one clock cycle. The stall is achieved by making the PTL sample its output value. To restart the execution after a stall of one cycle the

output of the PETF must assume the same value of the PTL, since the output of the multiplexer is connected to both PETF and PTL, at the rising clock edge after the error has been detected all the flip-flops will sample the correct value that is stored in the latches.

Channels control signals

Since in case of errors the communication channels at the input and output of the hierarchical block must be stopped, it is required to reset the control signals sent to them. Similarly to the enable of the registers, the control signals of each block are gated using the error signal of the same block. The gates required to stop the validation signals depend on the protocol used for the communication, in this thesis, channels from the Catapult synthesis library and from the MatchLib connections library have been tested, in both cases active high signals are used, so AND gates are inserted to reset them.

Glitch in error signal

One problem that comes from this Razor architecture is that, if after the rising clock edge the outputs of the PETF and PTL do not reach the XOR gate at the same time, there is the possibility that a glitch would propagate on the error line. Since this issue is present for all the Razor cells, the error signal of the hierarchical block can have more than one edge. Since this signal is connected to the enable of the registers this can lead, at best to an increase in power consumption, and at worst this can trigger the enable of the Razor cells and cause an erroneous sampling of the PETF.
To avoid these "glitches" an NTL is added along the error path in order to sample the error signal only when the high clock phase has ended. If a flip-flop was chosen as barrier, it would have implied that the propagation of the error must end before the falling clock edge, this is a strict timing constrain because the PTLs at the beginning of the error chain will be transparent until the falling clock edge. Using an NTL removes this constraint since it allows for time borrowing, it is however required to resolve the borrowing before the following rising clock edge to have a stable register enable signal. One issue that is generated by this latch is that now there is a loop in

the error path (starting from Razor PTL, error NTL, and back to the Razor PTL). It is a problem since when the device is started, the value stored in the error NTL is uncertain and it will cause the circuit to behave in an unpredictable way, to avoid this issue it is sufficient to add a reset signal to the error NTL.
A second issue is that the control signals for the channel can require more time than what is available (can be calculated with the constraint 3.3 ,but where T_{cycle} is equal only to the low clock phase due to the NTL barrier), for this reason the control signal are gated using the error signal bypassing the NTL.

3.2.2 Extension of the design flow

Due to the edits in the architecture of the error path, it is required to adjust the design flow accordingly.

Hierarchy preservation and enable signals

Considering that the different sections of the circuit will run concurrently, and it is required to stall only the section where an error has been detected without stopping the rest of the circuit, it is needed to work preserving the hierarchy of the design, at least until the netlist has been modified adding the Razor cells.
Since it is important to have a strict control over the enable signals that are used in the design the RTL produced by Catapult is requested without any enable signal. After a first synthesis, the enable mechanism is added by replacing the flip-flops in the design with ones with enable pins, and the control signal for the channels are provided with the logic required to stall them. This procedure is done only for the hierarchical block that the user has specified to be considered for the insertion of Razor.
Then, the detection of the setup critical endpoints is done on a block by block basis, again only the specified blocks are considered.

Insertion of Razor endpoints to setup critical paths

The procedure that is executed to insert the Razor mechanism in the design is described in the following pseudo code:

Algorithm 1 Insertion of Razor cells

Require: List of instances to check, list of registers to replace with Razor

```
1:
2: for all Instances in design to check do
3:     for all Setup critical FF in this instance do
4:         Replace FF with Razor cell
5:     end for
6:     if More than one Razor cell inserted in this instance then
7:         Create OR tree to merge errors
8:     end if
9:     Create NTL to filter out glitches
10:    Connect enable signal to all the registers
11:    Connect enable signal (bypass NTL) to all the interfaces
12: end for
13: Flatten hierarchy
```

The first step of the algorithm is to insert the Razor cells, the only difference from the previous implementation is that this step is done for each hierarchical block specified by the user, and that the logic gates required to form the multiplexer are created and connected to the enable signal.
In line 9 the NTL to eliminate the glitches and the logic to reset it are created. One of these latches is created for each hierarchical block where Razor is added.
Since the enable signals for the registers and the interfaces have been previously added to the design, now it is sufficient to connect these signals to the corresponding error signal (lines 10 and 11).
The last step is to flatten the design, this step is required by the Mix & Latch flow to work correctly.

Additional timing constraints

The constraints that have been added for this new architecture are:

- Constraining PTL feedback loop: the loop generated by the enable mechanism of each Razor cell represents an issue for the hold constraint because the output of the latch would reach its input before the end of the transparency phase. In normal conditions this can represent an issue since the value that should be stored will be overwritten by the one caused by the feedback. Looking at the

working principle of the error correction mechanism it is possible to understand that this condition does never occur, during normal operation the multiplexer will propagate the input coming from outside the razor cell, the signal from the feedback loop propagates to the PTL's input only the clock cycle after one error is detected, in that case it is correct that the output of the PTL reaches its input during the transparency phase.
It is however important to relax only the hold constraint for the loops that connect output and input of the same PTL and are used for the enable mechanism, because otherwise this will impact also loops that are external to the Razor cell. Since the multiplexer has been realized using three NAND gates as shown is figure 3.9, only the hold constraint for the loop that goes through the gate "1" has been relaxed, this way all the other loops can maintain their original constraints.

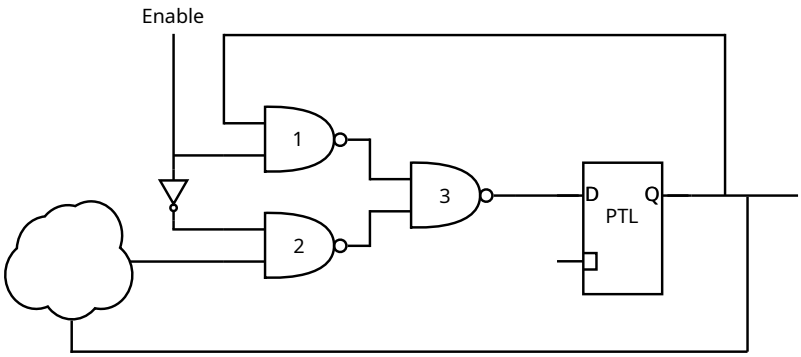

Figure 3.9: Logic used for the enable of Razor cells

- Maximum time borrowing for error NTL: to avoid that the error signal will change too close to the rising clock edge and cause setup violations due to the propagation as enable signal, the time borrowing on the NTL that filters the glitches is limited.

Extraction of timing information

In the original Mix & Latch flow the timing data was extracted assuming that only PTL were present as sequential elements, since this is no longer true, some edits are required to extract the timing data correctly. The changes required to consider the

PETF have already been discussed previously. Other changes have been made to consider the presence of NTLs, in particular: the amount of possible time borrowing on such elements is fixed to a value selected by the user, the arrival time for data that has been launched by the NTLs has been corrected to consider that the propagation starts on the falling clock edge, the insertion of NTLs to fix the hold violations has been prevented along the path that begin and end at NTLs used for the Razor mechanism.

Chapter 4

Results

The flow has been tested via post-layout simulations using a 28 nm FDSOI CMOS technology. To perform the logic synthesis from RTL code, Design Compiler from Synopsys has been used, then, Innovus from Cadence is used to perform place and route. For the circuits created with the SLD methodology, Catapult HLS from Mentor has been used to translate from high-level language to HDL code. The simulations have been performed using Questasim from Mentor.
More details on the design and testing methodology will be given for each test circuit.

4.1 Error detection

As circuit to test the error detection mechanism a simple 16-bit ripple carry adder (RCA) described in SystemVerilog has been used.

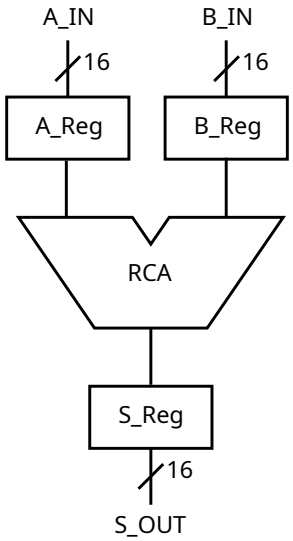

Figure 4.1: Schematic of the RCA

Optimization flow

To evaluate its impact on the results, different values for the setup slack threshold (from 0% to 30% of the clock period) are used to evaluate the endpoints to replace with Razor cells. The time borrowing allowed for the NTLs has been set to 25% of the clock period.

Threshold	Cell Count						
	PETF	PTL	NTL	Gates	Buff.+Inv.	Total	Increase
0%	48	0	0	122	89	259	0%
5%	48	2	2	148	68	268	3.47%
10%	48	2	5	148	52	255	-1.54%
15%	48	4	8	155	88	303	16.99%
20%	48	5	12	161	81	307	18.53%
25%	48	5	12	159	60	284	9.65%
30%	48	7	10	167	59	291	12.35%

Table 4.1: Number of elements in the RCA

Threshold	Cell Area (μm^2)						
	PETF	PTL	NTL	Gates	Buff.+Inv.	Total	Increase
0%	127.30	0	0	125.83	73.44	326.56	0%
5%	127.94	2.94	2.94	155.53	30.52	319.87	-2.05%
10%	126.64	2.94	7.34	143.62	24.81	305.35	-6.49%
15%	126.96	5.88	12.08	153.90	43.41	342.23	4.80%
20%	130.29	7.34	17.63	146.55	39.33	338.15	3.55%
25%	128.93	7.34	18.61	169.08	35.25	359.20	9.99%
30%	127.29	10.28	14.69	193.07	32.48	377.81	15.69%

Table 4.2: Area occupied by the different elements in the RCA

From the tables 4.1 and 4.2 it is possible to evaluate how the number of elements and their area has changed for the different setup thresholds. In the original design (threshold = 0%), only flip-flops are used as sequential elements in the circuit. After the insertion of the Razor the number of flip-flops does not change, but one PTL and an XOR gate are added for every endpoint that has been replaced with a Razor cell; if more than one endpoint has been added, OR gates are inserted to merge all

the errors signal into one.
All the endpoints that have been replaced are flip-flops that sample the output of the adder, since the time that is required to compute the sum increases from less significant bits (LSBs) to most significant bits (MSBs), if the setup threshold increases, Razor cells are naturally inserted starting from the MSBs and gradually replacing FF toward the LSBs.
After the Mix & Latch flow the hold violations have been resolved by inserting an amount of NTLs that increases with the number of PTL.
Due to the variability of the flow the area and number of elements in the design does not follow a monotone trend, but it is visible as an increase in Razor endpoints there is an increase in circuit area due to the additional latches and logic gates.

Simulation

To evaluate the results, all the layouts obtained at the end of the flow have been simulated. All the simulations have been run on the same sequences of inputs values (100K random values). For both the original circuit, and the one modified with Razor, the clock frequency has been swept from the nominal value, up to 1.7 times the nominal value.
A testbench has been realized in SystemVerilog to count the number of violations detected by Razor, the number of errors in the computation of the sum, and how many of that errors are on bits where Razor is not added. Errors cannot be detected by Razor either because there are violations in FF where the error detection has not been inserted, or because there are violations in the NTLs.
Since the metastability in the sequential elements caused by the increase of frequency is not addressed in this thesis, and that its presence will make the results unclear, the simulations are run in a way that eliminates the metastability. If the input of a cell changes near the clock edge that triggers the sampling (or that ends the transparent phase of a latch), the resulting output of the cell will always be the input value immediately before that edge, even if the setup or hold time are not meet. The presence of violations is nevertheless reported in the simulation logs, so it can be used to formulate additional considerations.

Preliminary considerations on the simulation As shown from the plot in figure 4.3 the original design will produce errors with a frequency increase of 12%. Since the critical path of the design has a setup slack close to zero, theoretically the circuit should fail even with a small increase of the frequency, this does not happen for two reasons: the avoidance of the generation of undetermined values in the simulation implies the removal of the setup times, this results in a higher working frequency before circuit's failure. The second reason is: due to the high number of input combinations, it has not been possible to find an input transition capable of causing the propagation along the entire critical path.
While the first cause of uncertainty cannot be removed, the second could be partially eliminated increasing the number of tested input combinations. Since the effect of Razor is visible also with these results, this level of uncertainty has been considered acceptable.

Error detection The increase in frequency at which the effect of Razor starts is visible in figure 4.2, here it is shown the number of times where the Razor mechanism has detected a change in the signal's value after the sampling of the PETS, in this condition the error signal is asserted. The frequency increase required to cause violations changes depending on the setup threshold that has been chosen, also in this case this is due to the effects explained before, because, if the critical path had been consistently stimulated, even a small increase in frequency would have caused the detection of the error at the setup critical endpoint were Razor has been inserted. It is nevertheless possible to see how an increase in threshold corresponds to a detection of errors at lower frequencies and a faster increase in detection count with the rise in frequency. This is because the shorter combinational paths are stimulated more frequently than the longer ones, when more flip-flops at the output of the adder are replaced with Razor cells, all the violations in the shorter paths can be detected and the uncertainty caused by the small number of input combinations is reduced.
The sudden drop in detections at high frequencies (from 50% increase and above) is caused by the impossibility of the input registers to correctly sample the inputs provided to the adder. This event is visible in all the plots and will be mentioned for all of them in their corresponding paragraph.

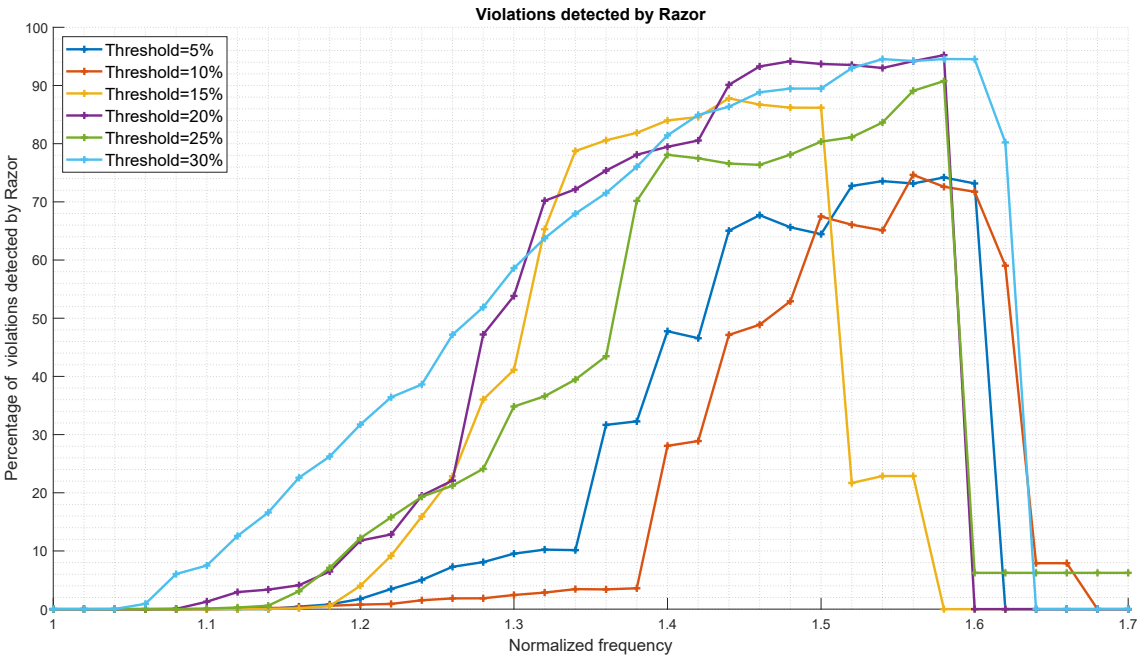

Figure 4.2: RCA simulation - Violations detected by Razor

Errors in the output value The number of times that a wrong value for the result of the addition has been detected, is displayed in figure 4.3. Here the trend is almost inverse to the one of the error detection, a higher number of endpoints replaced with Razor cells corresponds to a higher frequency increase before the generation of errors. This is simply because the output data of a Razor cell is sampled using the latch, and, due do the possibility of time borrowing, it is always correct until the constraint 3.1 is satisfied. As expected, in all the tested cases the detection of timing violations has been triggered before any error have effectively reached the output signals, the interval of frequency in which the violations are detected before any error occurs widens with the increase in setup threshold.

One important thing to study is from where those errors come from. For this reason, for every circuit that have been simulated, two types of error are counted and are traced in the plot. A first line indicates how many times there is an error in the addition's result, a second line indicates how many of those times there is an error also on the output values of FF that are not replaced by razor cells. An error that is not detectable by Razor can be caused by different things: error in sampling of the input registers, error in the sampling of the NTLs, error in the sampling of

output flip-flops that are not replaced with Razor. These two lines are sufficient to understand which element is responsible for the error.
For lower setup thresholds these two lines are almost overlapped for all the frequency range, the only difference is above a 50% increase where the constraint 3.1 is no longer satisfied, this means that all the errors are caused by a wrong sampling of the output bits where the error detection is not present. For the higher threshold, the two lines split apart at lower frequencies, in such cases not all the errors are on the lower bits, before the previously mentioned 50% increase limit, all the errors that are generated are caused by the early closing of the NTLs that has been explained in section 3.1.2, with higher thresholds the limit to the time borrowing becomes more challenging to satisfy and it causes a degradation of the performance for the error detection. At higher frequencies there is 100% error rate because of the error in sampling the inputs, the precise frequency from which this happens is not visible in this plot but it is known from the detection rate (4.2).

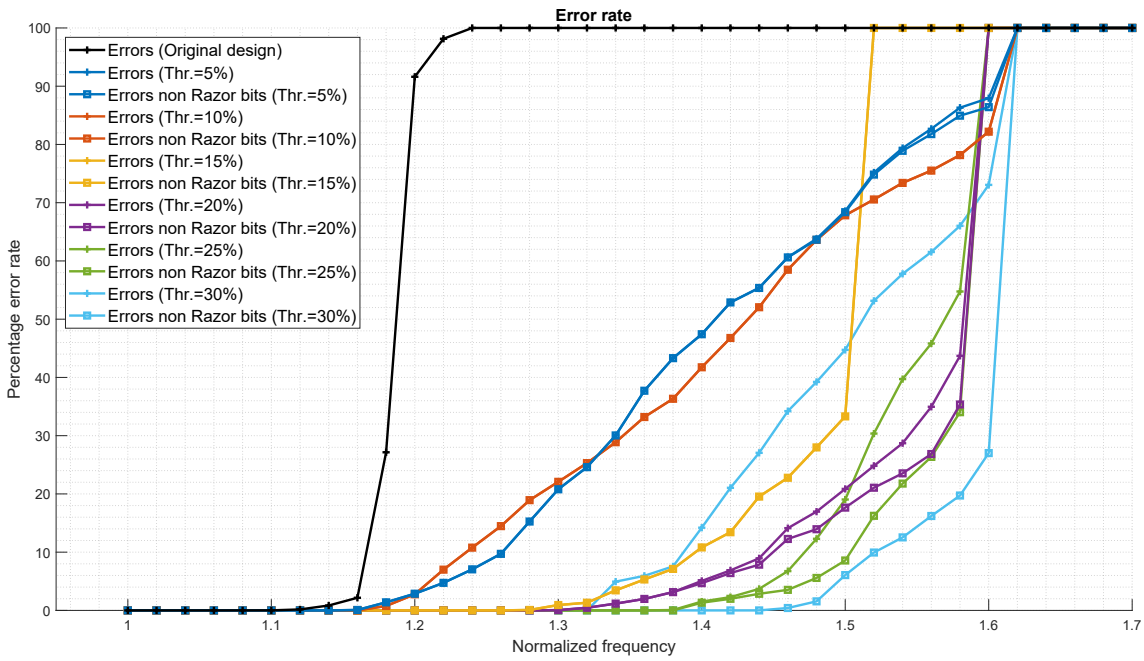

Figure 4.3: RCA simulation - Error rate

Power consumption To evaluate the power overhead caused by the insertion of Razor, during all the simulations that have been performed the activity of each

node of the circuit has been recorded and it has been used to calculate the power consumption.
Since in this circuit no error correction is present, the detection of errors does not cause changes in power consumption, so for all the simulations there is only a linear increase in power due to the higher frequencies. For every frequency point it is visible the power overhead caused by Razor, this overhead rises with the increase in setup threshold, this is because in those conditions a higher number of elements are present in the circuit (as displayed in the table 4.1). It is visible how there is a maximum power overhead equal to 27% compared to the original design. Also in this case, the error in the sampling of the inputs is visible, this time its effect is a drop in power consumption since in this condition no elements in the design are switching.

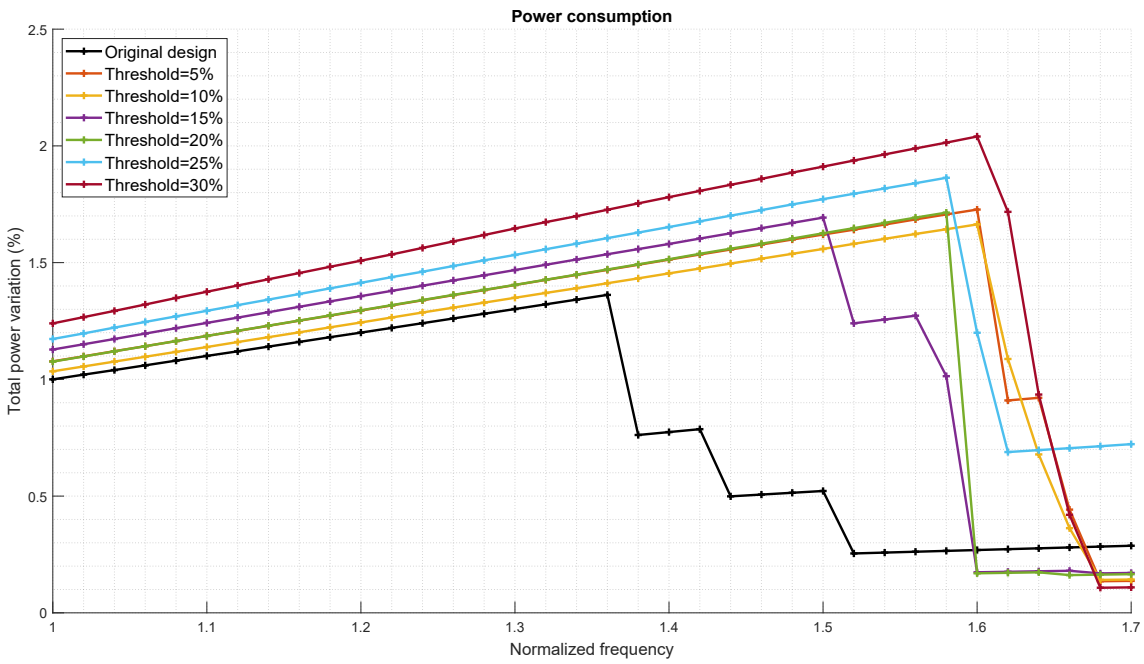

Figure 4.4: RCA simulation - Power consumption

The different contributions to the power have been separately calculated for the circuit working at nominal frequency without any error detected, they are then reported in the following table. The contribution that undergoes the biggest change is the leakage with a maximum increase of 60%, the following biggest increase is the one related to the switching of the capacitance with a 49% increase, finally, the

smallest increase is the one internal to the gates with an increase of 12%. Since the leakage power is only a minor part of the total power, the total increase in between the switching and internal power contributions.
Since this circuit is really simple, and with the higher thresholds the error detection is inserted to many of the FF, the overhead is substantial. A more reasonable amount of overhead is visible in more realistic test cases for this flow that are going to be analysed below.

Threshold	Powers (mW)				
	Internal	Switching	Leakage	Total	Increase
0%	1.014	0.666	3.023e-3	1.683	0%
5%	1.011	0.798	3.821e-3	1.813	7.72%
10%	0.968	0.769	3.453e-3	1.741	3.45%
15%	1.062	0.832	4.178e-3	1.898	12.77%
20%	1.051	0.756	4.012e-3	1.811	7.60%
25%	1.109	0.860	4.385e-3	1.974	17.29%
30%	1.141	0.991	4.846e-3	2.136	26.92%

Table 4.3: Power consumption of the RCA

4.2 Error detection and correction

To verify the error correction capabilities of the flow, two different designs have been exploited: a cascade of two finite impulse response (FIR) filters and a RISC-V processor. The two circuits have been realized starting from high level code (C++ and SystemC) and, using an HLS tool, a hardware description in Verilog has been generated. Both designs are organized in a dataflow manner and are latency-insensitive, but they differs in the type of communication channels between hierarchical blocks; the differences will be explained in detail in the corresponding sections.

4.2.1 FIR filters

The two filters have the same structure: they work on 18 bits samples and use two 18 bits coefficients, the output of the first filter is sent to a FIFO that is then connected

to the input of the second filter. At the input and output of the whole chain the same synchronization signals as the FIFO are used to manage the flow of data, a schematic of the design is displayed in figure 4.5. All the communication channels that are used have been taken from the Catapult logic synthesis library.

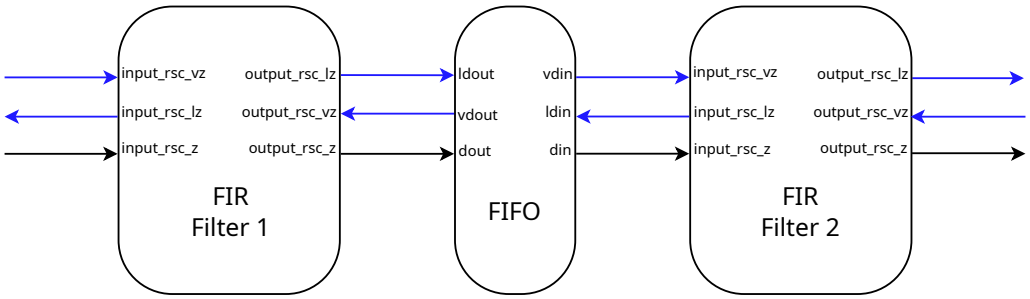

Figure 4.5: Schematic of the FIR filter circuit

Optimization flow

As for the RCA test case, different setup thresholds (from 0% to 30% of the clock period) for the insertion of Razor have been tested, the allowable time borrowing on the NTLs used to fix the hold violations is again equal to 25% of the clock period, time borrowing to the NTLs that filter glitches has not been allowed. The insertion of Razor has been permitted only in the two filters to avoid changes to the structure of the FIFO.

Threshold	Cell Count						
	PETF	PTL	NTL	Gates	Buff.+Inv.	Total	Increase
0%	491	0	0	2855	1554	4900	0%
10%	491	11	49	3355	1644	5550	13.26%
20%	491	33	125	3418	1524	5591	14.10%
30%	491	57	142	3522	1786	5998	22.41%

Table 4.4: Number of elements in the filters

Threshold	Cell Area (μm^2)						
	PETF	PTL	NTL	Gates	Buff.+Inv.	Total	Increase
0%	1599.58	0	0	4438.55	1087.54	7125.80	0%
10%	1592.43	17.14	72.63	5428.19	954.56	8065.02	13.18%
20%	1566.56	52.39	186.23	5502.94	923.55	8231.65	15.52%
30%	1542.82	84.71	210.39	5483.52	1131.63	8453.11	18.63%

Table 4.5: Area occupied by the different elements in the filters

As for the RCA case with an increase of the number of Razor cells that are inserted there is an increase in the area occupied by the circuit. In this case there is a big increase (13%) of area when the first Razor cells are inserted. Since the following increases caused by additional error detection cells are more gradual, this first jump is probably due to the insertion of the error correction mechanism. In this case the area occupied by the PETF decreases with the increase of inserted Razor cell, this is caused by the type of cells that are used. The standard FF are equipped with a mechanism to manage the enable, while the PETF in the Razor cells do not have an internal enable logic since they use an external multiplexer.

Simulation

Since the circuit has been realized using HLS, an automated flow to simulate the circuit is provided. This flow, starting from C++ code of the test procedure, automatically generates a tetstbench capable of handling the I/O interfaces to the supply the input and read the outputs of the circuit at the correct times, it also generates the scripts required to run the simulation software (also in this case Questasim from Mentor).
As for the previous case, all the simulations have been run with the same sequence of input samples (1K random values), while the coefficients for the filters are kept constant; the metastability has been again not considered in the simulations.

Preliminary considerations on the simulation To visualize the results two different plots are generated, if figure 4.6 it is displayed the percentage of samples where some error in the computation has occurred, unlike the RCA case, here it is not possible to easily detect the cause of the error. A second plot (figure 4.7)

shows the variation in the amount of time required to complete the simulation, with the increase in clock frequency this time should ideally decrease going towards zero, but, since the error correction implies the addition of a stall, when the penalty of recovering from the errors becomes greater than the advantage of a higher frequency, the simulation time will start to increase.
As the plot in 4.6 shows, in the original circuit no errors occur until the frequency is increased by 12%, also in this case the sources of this uncertainty are the same as the RCA (small number of input combinations, removal of undetermined values). Nevertheless, the effect of Razor can be correctly appreciated.
To evaluate only data that is meaningful, not all the result of the simulations have been inserted in the plots: for both the simulation times (4.7) and energy consumption (4.8), the results are reported only up to the point where the first error occurs.

Error detection By looking at the trend of the simulation times, it is possible to see how all the designs follow the ideal line up to a 10% increase, at that point errors are generated in the design without Razor. In all the other designs no errors are detected until the frequency is increased by at least 16%. After that point it is possible to observe how the circuits can work without generating errors, but with an increasing number of stalls.
Contrary to the RCA case, the frequency from which the detection of timing violations starts does not change much depending on the setup threshold, but it is still true that a higher threshold causes a higher interval where the detection occurs and no errors are committed by the circuit.
Increasing the frequency further causes the generation of errors also in the designs with Razor, inspecting the causes of the circuit's failures by looking at the simulator logs it emerged that: for the design with threshold 10% the errors are caused by the endpoints that are not replaced by Razor cells, instead, for the other two designs, the errors are caused by the sampling of the NTLs. This behavior where a higher setup threshold causes the point of failure to become the latches placed to fix the hold violations is consistent to what seen in the case of the adder.

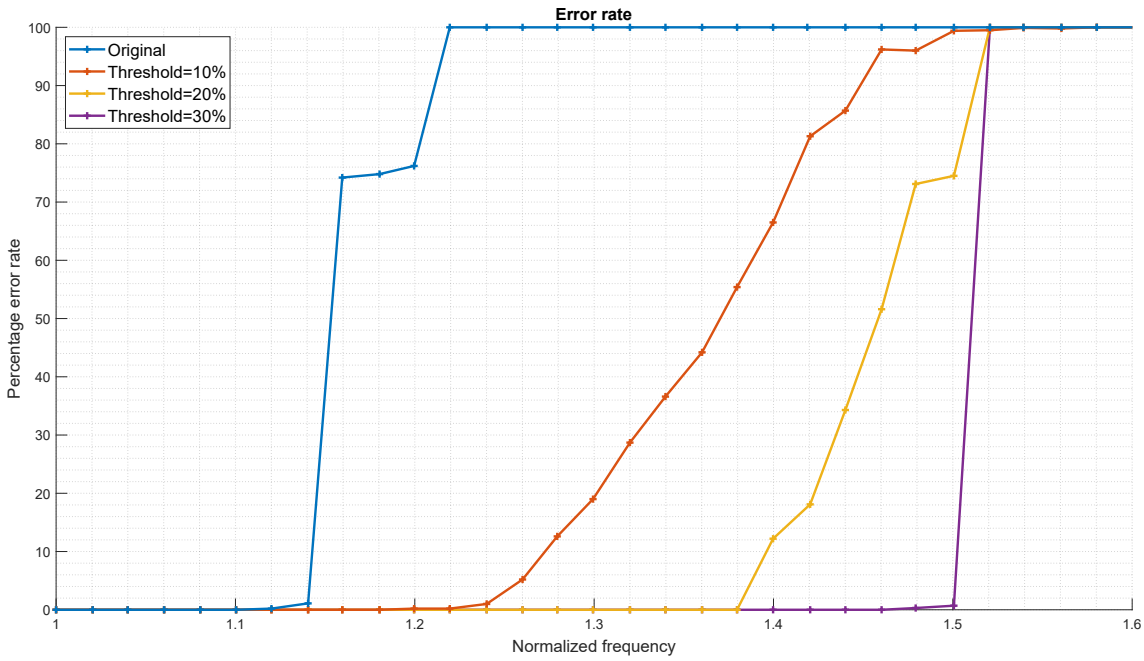

Figure 4.6: FIR simulation - Error rate

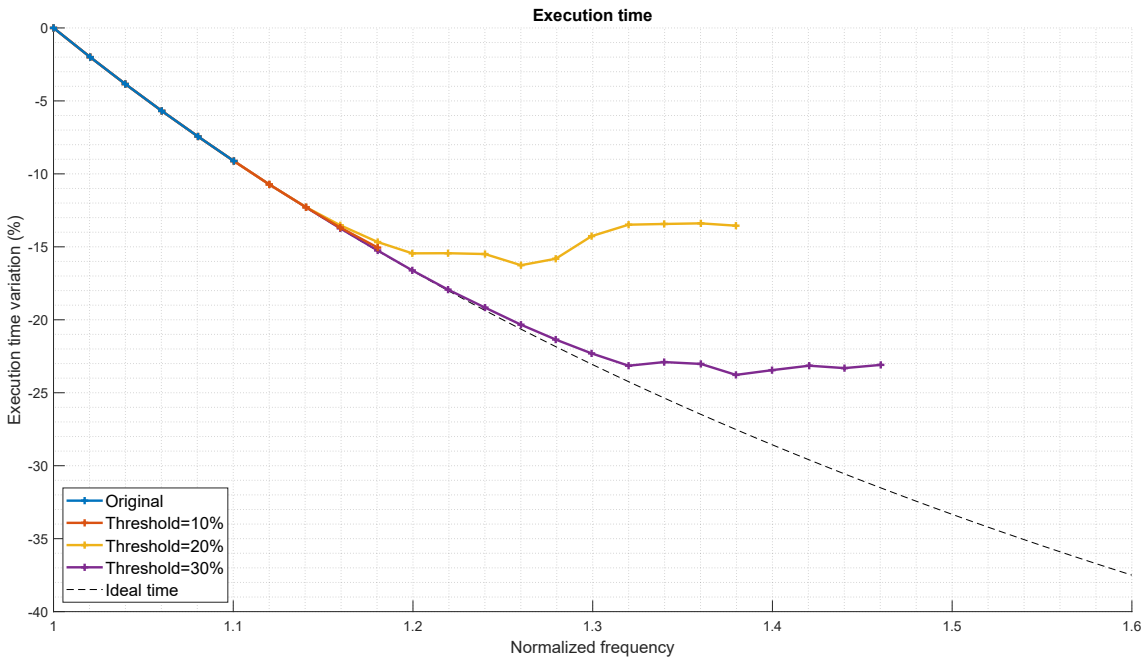

Figure 4.7: FIR simulation - Execution time

Power consumption Also for this design the power consumption has been computed studying the activity of the circuit. Since in this case the time required to terminate the simulation is important, due to the insertion of stall cycles to correct the errors, the power consumption is multiplied by the simulation time to compute the energy that the circuit has consumed.
With the increase in clock frequency, it is visible how initially the energy remains constant since the power consumption increases but the time required to conclude the simulation decreases. When errors start to being detected, the number of stalls increases accordingly, causing the increase in total energy.

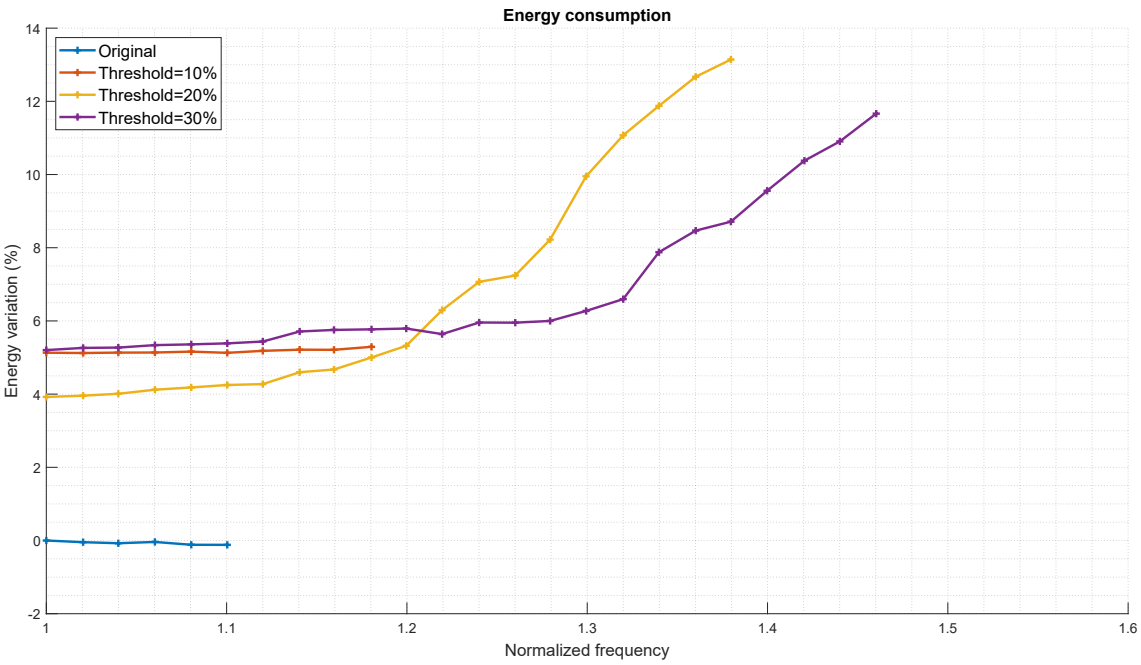

Figure 4.8: FIR simulation - Energy consumption

Also in this case the different contribution of the power consumption are evaluated at nominal frequency and in absence of errors. The increase for the different power contributions reflects what previously seen in the RCA case, but, the maximum overhead is reduced compared to the previous case since a reduced number of endpoints has been replaced.

Threshold	Powers (mW)				
	Internal	Switching	Leakage	Total	Increase
0%	6.05	7.84	0.14	14.03	0%
10%	6.37	8.20	0.18	14.75	5.13%
20%	6.07	8.21	0.16	14.43	2.85%
30%	6.46	8.12	0.19	14.76	5.20%

Table 4.6: Power consumption of the filters

Voltage scaling As stated in the introduction of this thesis, a major issue for digital circuits is the power dissipation, and that Razor is a valid solution to this problem. To evaluate the possible advantages that Razor can bring, a different type of simulation is executed.

To reduce the power consumption a solution that can be used in conjunction with Razor is dynamic voltage scaling, in this technique the voltage supplied to the circuit can be decreased in order to reduce the power consumption. However, with a lower voltage, the speed of the gates is reduced, and if no countermeasures are taken this will cause an erroneous behavior of the circuit. Razor is a valid way to monitor that the reduction in the circuit's speed does not introduce unrecoverable errors.

To check the effect that the reduction of voltage has on the behavior of the circuit, the delays of all the gates and interconnections in the final layout of the circuit are calculated using the data from different library corners. These different libraries have the same process corner and operating temperature condition, but have a different voltage supplied to the cells.

Due to the limited availability of different operating points for the library, it has been possible to simulate only for three different supply voltages. The design flow has been executed for a voltage of one volt, so this is considered as the nominal condition. At the end of the design flow, the timing parameters have been extracted also for 0.9 V and 0.8 V.

The simulations have been executed using the same procedure of the previous ones but the frequency is not swept, instead it is fixed at the nominal one.

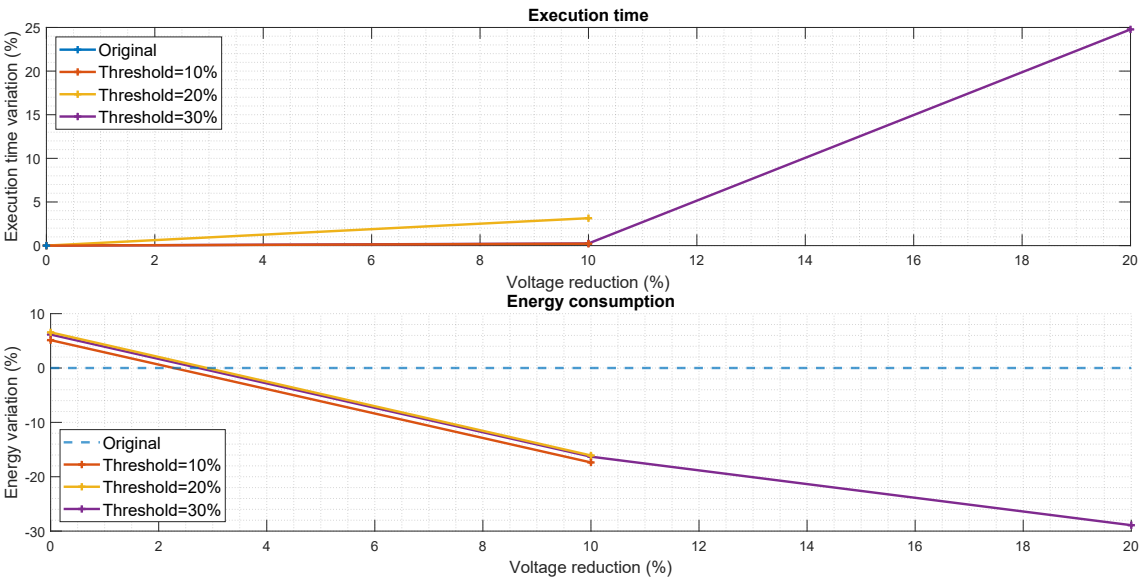

Figure 4.9: FIR simulation - Voltage scaling

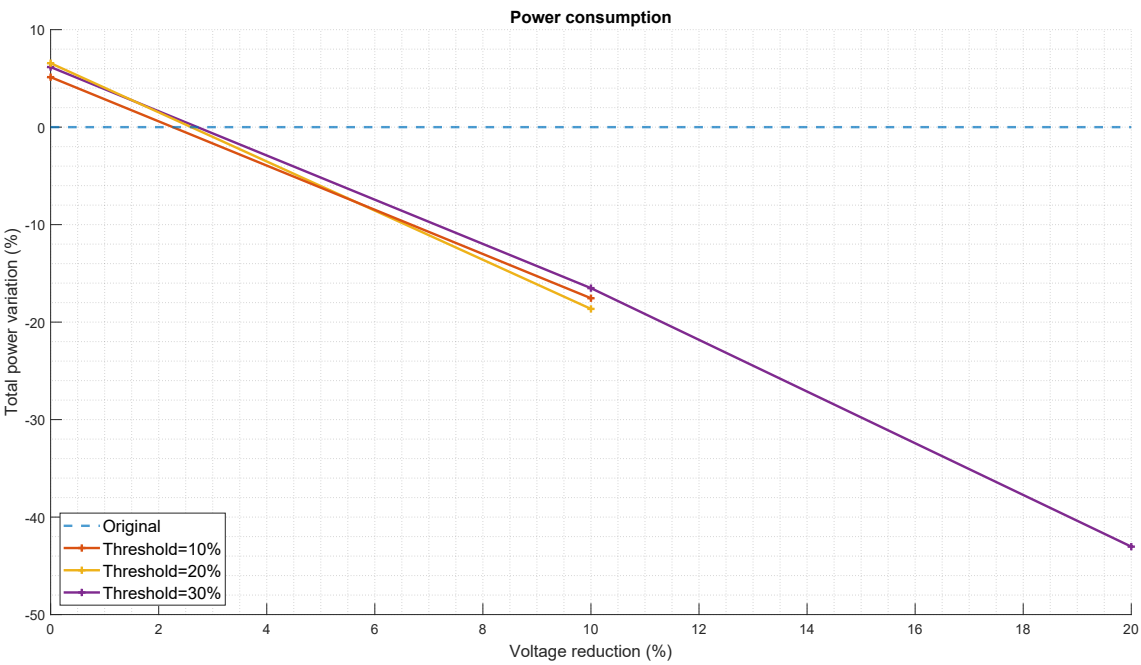

Figure 4.10: FIR simulation - Power consumption

To evaluate the results, two plots that display the same variations to the ones in figure 4.7 and 4.8 have been produced, but this time on the horizontal axis the

voltage is shown instead of the frequency. This time no plot is produced for the count of errors since, if even an error is generated, the results would become meaningless. Instead, only the data points of the simulation where no errors have been detected are inserted in the plots.
The effect of a reduction in voltages should be the same as an increase in frequency and, even from the small number of data points, this is visible as an increase in simulation time due to the addition of stall cycles to correct errors. Instead of an increase in performance the goal of the voltage scaling is a decrease in power consumption, from figure 4.10 it is visible how even with a small reduction in voltage it is possible to have a reduction in power much bigger than the overhead introduced by Razor.
On first site these results can seem underwhelming, at best a 40% reduction in power with a reduction of 25% in speed, since power consumption depends quadratically with voltage while speed depends linearly (at least when supply voltage is higher than the threshold voltage) a 25% decrease in speed should ideally be met with a 50% decrease in power. It is important to notice that the case shown by the simulation represents the worst possible case, in real condition in most of the cases there will be a reduction in power without any reduction in performance, this is because in a real circuit not all the transistor in the design will behave like the slowest process corner.

4.2.2 RISC-V processor

As second test circuit for the error correction mechanism an open source RISC-V core has been used.
The processor that has been chosen is called DRIM4HLS, it is a high-level description in SystemC of an in-order 32-bit RISC-V processor based on the HL5 core from Columbia University. LI channels have been used to exchange data between the different stages of the pipeline and to communicate with the memories. In this case the channels are simple combinatorial signals taken from the MatchLib Connections library. The instruction and data memory are not part of the processor, they are only used for simulation.

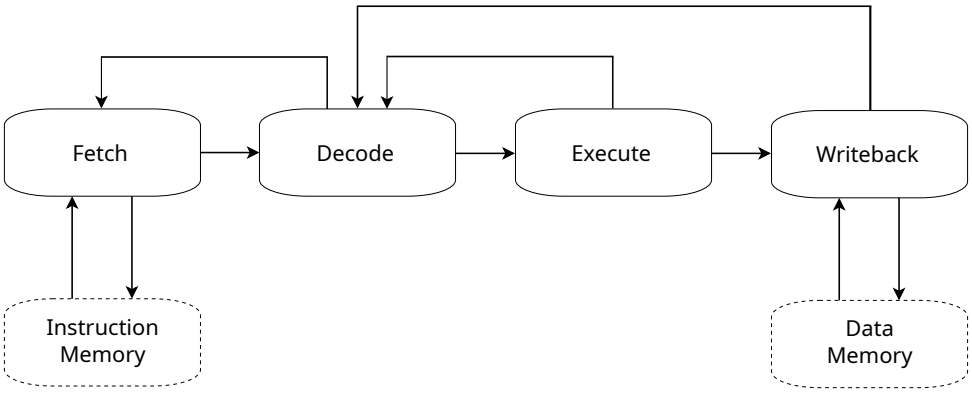

Figure 4.11: Schematic of the RISC-V processor

Changes made to the core

Unfortunately, Razor cannot be applied to the processor as it is. This is because, while it is true that the processor is LI, it can tolerate additional latency only if it is generated from specific operations or from the memories, this means that if latency is added by the error correction mechanism of Razor this will ruin the execution.
The cause of this issue is the way some of the channels are read. All the blocks that receive a feedback from a stage later in the pipeline read the communication channel in a non-blocking manner, this means that they will continue the execution even if no data is available in the channel. An example is the fetch stage, if no data is available from the feedback coming from the decode stage, the program counter will be incremented to continue sequentially in the process execution. Otherwise, if data is available, a jump to the memory address specified by the decode stage is performed. This implies that, if in the design with Razor a stall is inferred in the decode stage, it can cause the execution of the jump operation to not be correctly performed.
To fix this issue, all the non-blocking read have been replace with blocking operations. Moreover, checks have been added to avoid deadlock at the beginning of the execution.
One advantage of the style of communication used previously was that, in some cases, it is possible to reduce the power consumption by avoiding sending data to following pipelining stages if no operations are requested. For example, if in the decode stage a jump instruction is requested, no data will be sent to the execute stage and the execution will go on without unnecessary activate the following pipeline

stages. Unfortunately removing the non-blocking operations do not permit this functionality, to avoid deadlocks it is necessary that all the stages will propagate data even if no operations are required.
The edits brought to the design will degrade the performances of the processor; while this degradation can be reduced with an improved processor design it is now overlooked since the interest of this work is only to evaluate the impact of the addition of Razor to the device.

Optimization flow

Since this last test circuit is several times more complex that the previous ones, in this case, to limit the time required to execute the flow and to simulate the final circuit, only one setup threshold has been evaluated for the insertion of the error detection. Since from the FIR filters case the setup threshold that provided the best result is the 30% one, this is used also for this design. The same timing constraints have also been used (time borrowing on the NTLs added by Mix & Latch equal to 25% of the clock period, time borrowing to the NTLs that filter glitches not allowed).

Threshold	Cell Count						
	PETF	PTL	NTL	Gates	Buff.+Inv.	Total	Increase
0%	6755	0	0	19829	11676	42075	0%
30%	6755	246	245	21206	15491	44447	5.64%

Table 4.7: Number of elements in the processor

Threshold	Cell Area (μm^2)						
	PETF	PTL	NTL	Gates	Buff.+Inv.	Total	Increase
0%	23817	0	0	22547	8039	54403	0%
30%	23537	361	362	27591	9859	61710	13.43%

Table 4.8: Area occupied by the different elements in the processor

With the parameters that have been chosen to run the flow all the setup critical endpoints have been found in the execute stage. In this case 246 out of the 6755 flip-flops that form the design have been replaced with Razor cells, considering all the other logic cells required to fix the hold violations and to create the error correction mechanism an area overhead of 13% can be observed.

Simulation

Also for this last design the performances of Razor have been evaluated by simulating the circuit with different clock frequencies. This has been done for three different software benchmarks to see if it can influence the impact that the stalls caused by the error correction have on the execution time.
The three software that are executed are: fast Fourier transform (FFT) of 1024 samples, multiplication of two 256 x 256 arrays, sorting (using the quicksort algorithm) of a list of 512 values. These software are selected as benchmarks since they require arithmetic computations that should stimulate the paths where the Razor cells have been inserted. The programs have been written in C language and compiled using the standard RISC-V GNU toolchain.
In the original testbench provided with the core no latency is assumed for the two memories, instead, a random latency is applied for each access. This random behavior would make the reading of the results unclear, to eliminate this uncertainty a fixed latency of two clock cycles is applied for the instruction memory and a latency of five clock cycles is applied for the data memory. Whit this change, the only element in the design that can cause an increment in the simulation time is the error correction mechanism.
Unfortunately, due to the high complexity of the design it was not possible to consistently stimulate the generation of timing violation by the critical paths. To still be able to evaluate the performance penalty of the error correction, the generation of stalls has been manually forced during simulation. The rate at which the stalls are inferred has been chosen by inspecting the results obtained from the previous test cases and from the trend that the timing violations have in publications regarding this topic. After these considerations, it has been chosen to generate timing violations with a rate that increases exponentially with the increase in clock frequency, reaching a stall rate of 100% (one stall for every clock cycle) at a frequency increase of 30%. Such trend is shown in figure 4.12, the point corresponding a 30% frequency increase has not been simulated because it would have required a considerable amount of time since with an exponential increase in stalls also the time required to terminate the simulations increases exponentially.

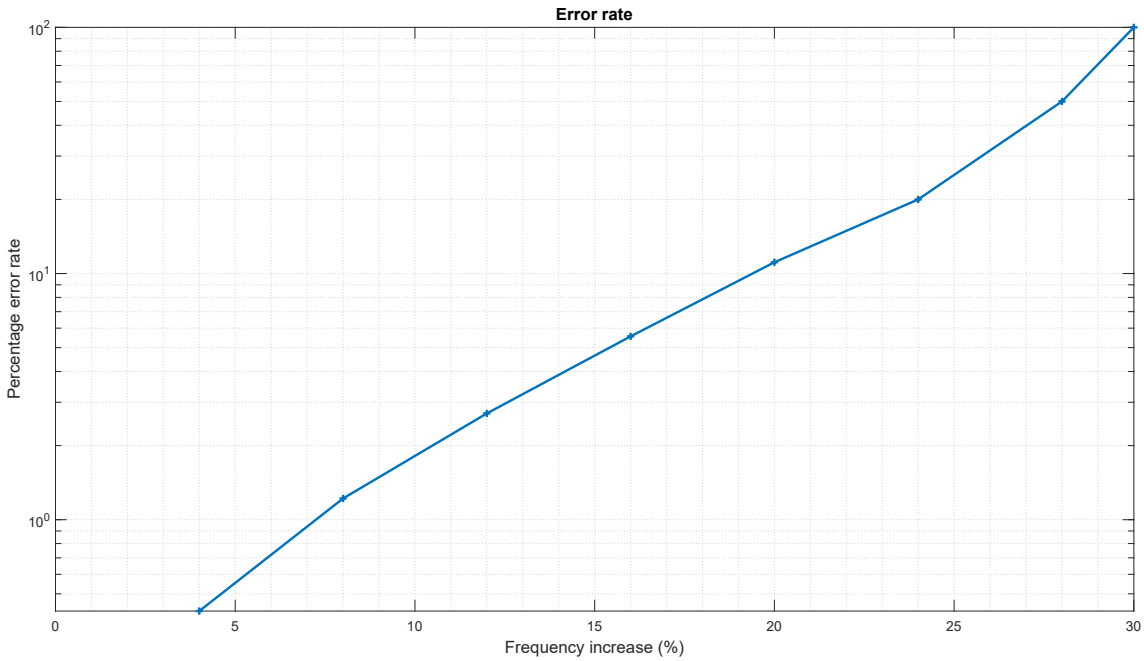

Figure 4.12: RISC-V simulation - Error rate

Error detection Since the simulation time required to terminate the execution is different for each of the three benchmarks and it is not important to analyze the performance of Razor, it has been normalized to the time used to terminate the execution at the nominal clock frequency, it is than shown in figure 4.13 to see how the stalls impact the performances.

The results are consistent from what observed from the FIR test case, the simulation time initially reduces due to the increase in clock frequency. When the number of timing violations rises, the penalty introduced by the error correction is greater than the benefits of increased frequency, so the simulation time stops reducing and later it rises back.

It is possible to notice that the reduction in throughput caused by the stalls is not equal in magnitude with the number of stalls. For example, looking at the right most point all the software require 30% more time to terminate compared to the ideal completion time, but this happens when a stall is inferred for 50% of clock cycles. The difference between stall rate and throughput reduction in this case is caused by the number of cycles where the processor is waiting to receive data from the memories, because in such case a stall in the execution stage will be masked by

the stall caused by the I/O operation. This confirms that the Razor mechanism can be successfully used in LI applications to reduce the performance penalty that is introduced by the error correction mechanism.

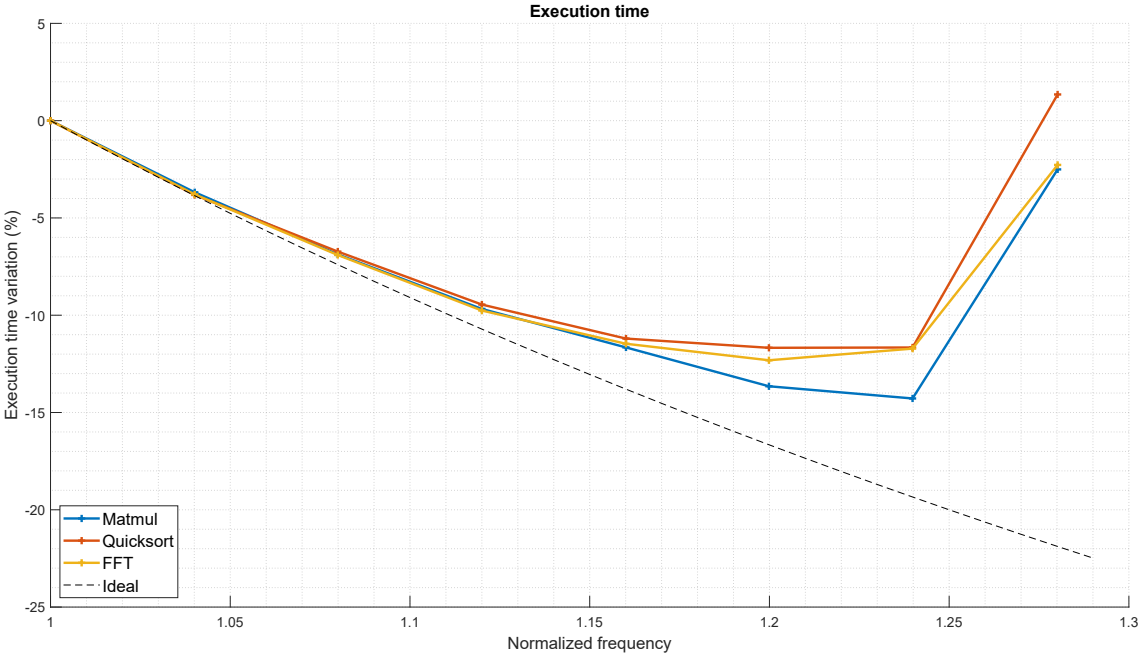

Figure 4.13: RISC-V simulation - Execution time

Power consumption Evaluating the power consumption for the three benchmarks, and multiplying it with the matching simulation time, the dissipated energy has been computed. To eliminate its dependence from the software that has been run, it is then normalized to the energy dissipated to terminate the execution at nominal clock frequency.

Since the errors are manually asserted at the end of the error path, the trend of the consumed energy is almost the same as the one of the simulation time. It is again possible to see the reduction of overhead caused by the LI nature of the design.

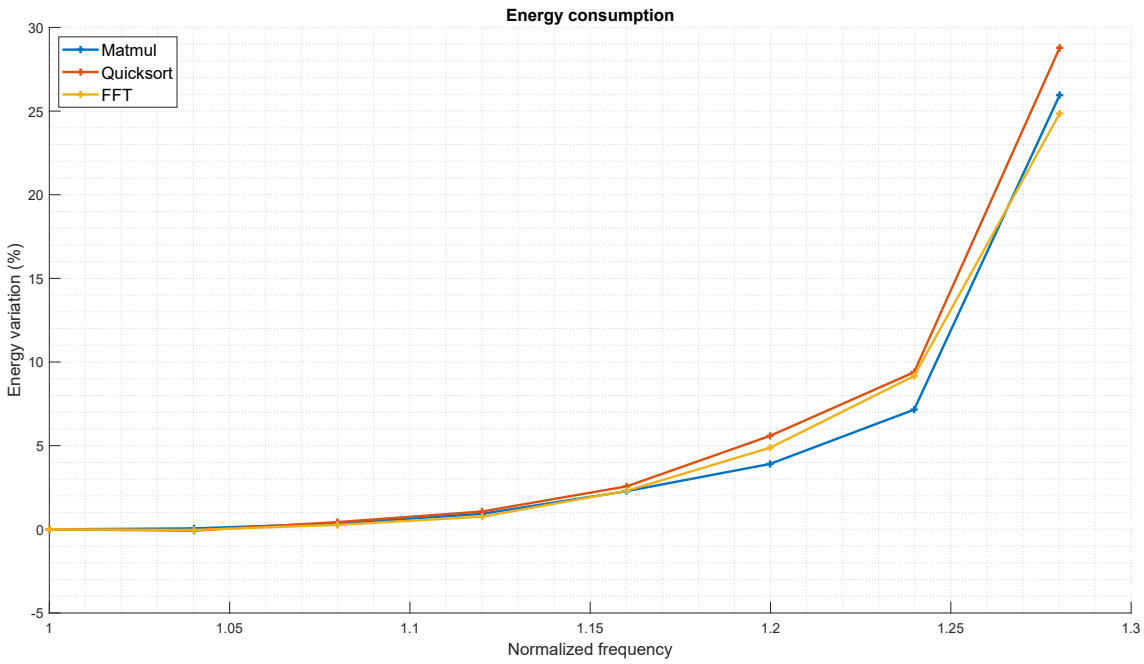

Figure 4.14: RISC-V simulation - Energy consumption

In the following tables (4.9, 4.10, 4.11) are displayed the different power contributions for both the original circuit and the one where Razor has been inserted. It is possible to see that the power consumption does not change by much depending on the software, a maximum variation of 4% is observed. The introduction of Razor to the circuit has only a small impact on the power, a maximum of 7% increase in power consumption can be observed from the three cases.

Threshold	FFT (Powers in mW)				
	Internal	Switching	Leakage	Total	Increase
0%	11.06	7.69	0.74	19.49	0%
30%	11.30	8.07	0.88	20.25	3.89%

Table 4.9: Power consumption for the FFT benchmark

Threshold	Matrix Multiplication (Powers in mW)				
	Internal	Switching	Leakage	Total	Increase
0%	10.73	6.95	0.74	18.42	0%
30%	10.93	7.28	0.87	19.08	3.58%

Table 4.10: Power consumption for the matrix multiplication benchmark

Threshold	Quicksort (Powers in mW)				
	Internal	Switching	Leakage	Total	Increase
0%	11.02	7.61	0.74	19.37	0%
30%	11.25	7.94	0.88	20.06	3.56%

Table 4.11: Power consumption for the quicksort benchmark

Voltage scaling Following the same procedure as the FIR filters it is possible to simulate the possible power gains achievable with voltage scaling. The three benchmarks have been run on the processor equipped with Razor at nominal frequency with supply voltage equal to 1, 0.9 and 0.8 volts. The same benchmarks have been run on the original processor at supply voltage 1 V to evaluate the power overhead caused by the insertion of Razor.

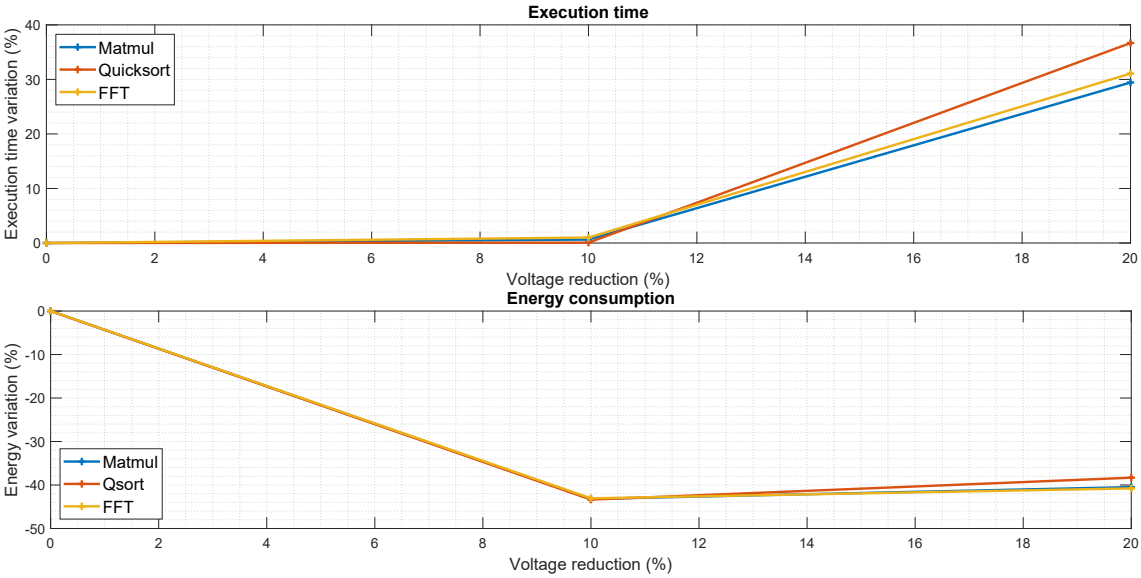

Figure 4.15: RISC-V simulation - Voltage scaling

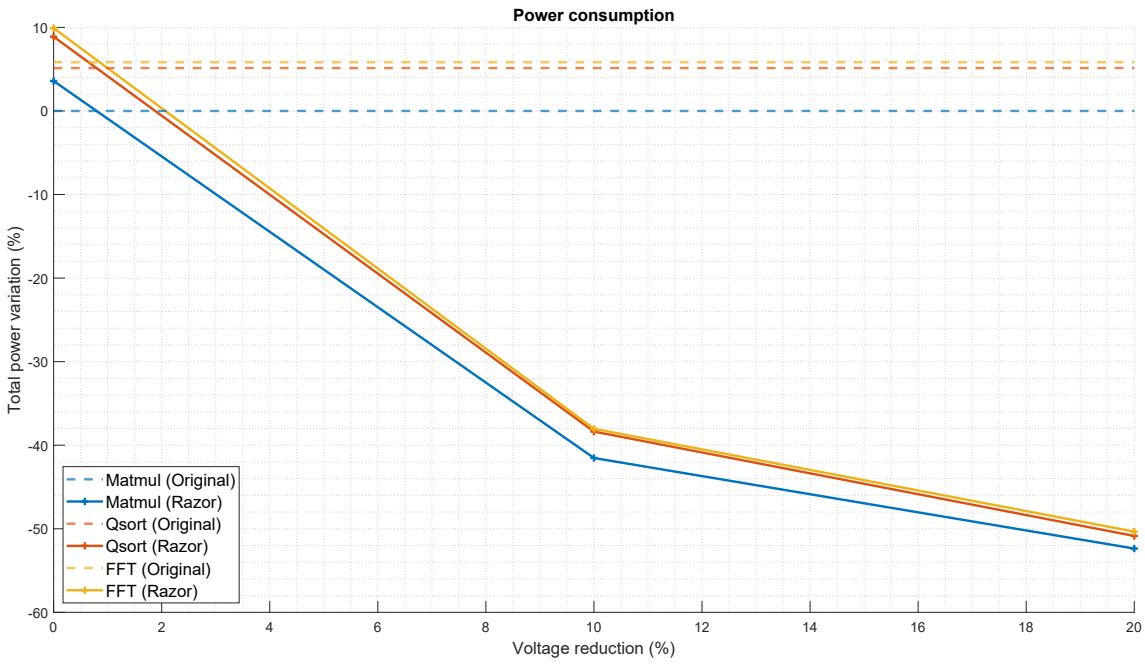

Figure 4.16: RISC-V simulation - Power consumption

With a voltage reduction of 10%, from the STA, a slowdown of 13% is observable, while for a reduction of 20% the circuit is 40% slower than in nominal condition. By using again the same model in 3.7, the error rates are equal to 3% and 100% respectively for 0.9 V and 0.8 V.

The trend of the plots in figure 4.15 is similar to the one of the filters. It is again possible to see how the simulation time does not increase as much as the amount of stall cycles (there is a 37% decrease in performance with a stall rate of 100%). This time, contrary to what seen for the FIR filters, since the amount of stalls is much higher, it is possible to see how a bigger reduction in voltage does not mean a reduction in energy because of the penalty caused by the stalls.

Chapter 5

Conclusions and Future Work

In this thesis work the Razor technique has been adopted in non-traditional methods to improve performance and energy efficiency. An automated flow has been developed to transform a traditional flip-flop design to an error-tolerant one that can be deployed in conjunction with frequency scaling and dynamic voltage scaling. A novel approach has been used to deal with the timing issues caused by the double sampling of the signals, latches transparent on the negative clock level are used to avoid early arrival of data to the Razor cells at the end of the setup critical paths. A new methodology has been tested to deploy the Razor mechanism to latency-insensitive circuits. The compartmentalization of the circuit in smaller blocks of logic unlocks potential for a low performance overhead error recovery that is achieved by a stall of one clock cycle only in the block where the error is detected, while the rest of the circuit can continue without halting. This error recovery is also scalable to bigger or aggressively clocked designs, since it does not rely on a global stall signal that would degrade the maximum working frequency.

The results that have been observed from the simulations of the filters confirm that the flow has been capable of automatically introduce the Razor error correction and correction mechanism to the circuits. From the simulation of the processor is instead evidenced that the application of Razor to latency insensitive designs can reduce the performance penalty introduced by the error correction. Finally, for both of the circuits, the possible performance gains obtainable with frequency scaling, and power consumption improvements achievable with voltage scaling have been proven.

This work can be improved in different ways: the metastability of the error signal should be investigated and its threat to the functionality of the circuit eliminated, the area occupation and power consumption after the insertion of Razor can be further optimized, more thorough testing on the failing of the critical paths to reduce the uncertainty in the results should be made.

Bibliography

[1] Jörg Henkel et al. "New trends in dark silicon". In: *Proceedings of the 52nd Annual Design Automation Conference*. 2015, pp. 1–6.

[2] K. Bernstein et al. "High-performance CMOS variability in the 65-nm regime and beyond". In: *IBM Journal of Research and Development* 50.4.5 (2006), pp. 433–449. DOI: 10.1147/rd.504.0433.

[3] Sani R Nassif. "Design for variability in DSM technologies [deep submicron technologies]". In: *Proceedings IEEE 2000 First International Symposium on Quality Electronic Design (Cat. No. PR00525)*. IEEE. 2000, pp. 451–454.

[4] Keith A Bowman et al. "A 45 nm resilient microprocessor core for dynamic variation tolerance". In: *IEEE Journal of Solid-State Circuits* 46.1 (2010), pp. 194–208.

[5] D. Ernst et al. "Razor: a low-power pipeline based on circuit-level timing speculation". In: *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.* 2003, pp. 7–18. DOI: 10.1109/MICRO.2003.1253179.

[6] Filippo Minnella et al. "Mix & Latch: An Optimization Flow for High-Performance Designs With Single-Clock Mixed-Polarity Latches and Flip-Flops". In: *IEEE Access* 11 (2023), pp. 35830–35840. DOI: 10.1109/ACCESS.2023.3265809.

[7] KA Sakallah, TN Mudge, and OA Olukotun. "optimal Clocking". In: *Ann Arbor* 1001 (1990), pp. 48109–2122.

[8] Luca P. Carloni et al. "A Methodology for Correct-by-Construction Latency Insensitive Design". In: *The Best of ICCAD: 20 Years of Excellence in Computer-Aided Design*. Ed. by Andreas Kuehlmann. Boston, MA: Springer US, 2003, pp. 143–158. ISBN: 978-1-4615-0292-0. DOI: 10.1007/978-1-4615-0292-0_12. URL: https://doi.org/10.1007/978-1-4615-0292-0_12.

[9] James Tschanz et al. "Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance". In: *2009 Symposium on VLSI Circuits*. 2009, pp. 112–113.

[10] David Blaauw et al. "Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance". In: *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*. 2008, pp. 400–622. DOI: 10.1109/ISSCC.2008.4523226.

[11] Matthew Fojtik et al. "Bubble Razor: Eliminating Timing Margins in an ARM Cortex-M3 Processor in 45 nm CMOS Using Architecturally Independent Error Detection and Correction". In: *IEEE Journal of Solid-State Circuits* 48.1 (2013), pp. 66–81. DOI: 10.1109/JSSC.2012.2220912.

[12] Keith A. Bowman et al. "Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance". In: *IEEE Journal of Solid-State Circuits* 44.1 (2009), pp. 49–63. DOI: 10.1109/JSSC.2008.2007148.

[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)*. 2013, pp. 1–2. DOI: 10.1109/S3S.2013.6716531.

[14] Huimei Cheng et al. "Converting Flip-Flop to Clock-Gated 3-Phase Latch-Based Designs Using Graph-Based Retiming". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.4 (2022), pp. 979–992. DOI: 10.1109/TCAD.2021.3068109.

[15] Vivek Kozhikkottu, Sujit Dey, and Anand Raghunathan. "Recovery-based design for variation-tolerant SoCs". In: *Proceedings of the 49th Annual Design Automation Conference*. 2012, pp. 826–833.

[16] Yuankai Chen, Xuan Zeng, and Hai Zhou. "Recovery-based resilient latency-insensitive systems". In: *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2014, pp. 1–6.