A Synthesis Toolflow for the Predictable Implementation of High-Performance Bundled-Data Asynchronous NoCs on FPGA

Giuseppe Chessa
Department of Computer Science
University of Manchester
Manchester, UK
giuseppe.chessa@manchester.ac.uk

Elena Bellodi
Dipartimento di Ingegneria
Universitá di Ferrara
Ferrara, Italy
elena.bellodi@unife.it

Michele Favalli
Dipartimento di Ingegneria
Universitá di Ferrara
Ferrara, Italy
michele.favalli@unife.it

Davide Zoni
Dipartimento di Elettronica Informazione e Bioingegneria
Politecnico di Milano
Milano, Italy
davide.zoni@polimi.it

Davide Bertozzi
Department of Computer Science
University of Manchester
Manchester, UK
davide.bertozzi@manchester.ac.uk

***Abstract*—FPGA implementation is essential for verifying asynchronous NoCs and validating design requirements. However, prototyping these circuits presents challenges in preserving timing integrity due to mismatches with FPGA timing models. The adoption of bundled-data NoCs with one-sided relative timing constraints further complicates their high-performance mapping. Current CAD flows focus on correctness, but performance optimization is hindered by poor control over timing convergence, leading to overdesigned margins and wasted performance. This paper proposes a methodology that tightly controls relative timing margins through selective net rerouting and delay constraint tuning. On an Artix-7 FPGA using Vivado, the implemented asynchronous NoC switch achieves 40% lower latency and up to 75% lower energy-per-packet than a synchronous counterpart.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Asynchronous networks-on-chip (NoCs) use handshaking protocols instead of a reference clock, offering advantages in area, performance, and power by eliminating clock distribution overhead, enabling activity gating during idle times, and simplifying large-scale physical design [1]. This makes them a key enabler for largely integrated 2.5-D or 3-D many-core processors [2] and a well-matched communication backbone to the requirements of emerging applications, especially neuromorphic computing [3]–[5] and smart sensors for edge computing or robotic control [6], [7].

NoC synthesis follows standard digital design steps, often using FPGAs for rapid prototyping and even for final implementation. FPGA prototyping is especially important for asynchronous NoCs, helping designers gain confidence in clockless communication architectures. However, mainstream FPGA verification is inefficient for asynchronous circuits due to a mismatch between handshaking protocols and global clock synchronization. Additional challenges include hazards in synchronous FPGAs, limited timing access, and a lack of resources like arbiters, delay lines and C-elements.

Extensive research has been carried out to bridge the gap between asynchronous circuits and FPGA hardware. While radical approaches have explored asynchronous FPGA architectures [8], [9], these are not commercially available and lack dedicated toolchains. An alternative is synchronous emulation [10], [11], where a synchronous model of the asynchronous circuit is implemented on FPGAs, preserving functionality and timing constraints. This strategy focuses on functional equivalence and FPGA mapping, rather than high performance or energy efficient implementations.

To work around this limitation, direct mapping frameworks have been developed, where asynchronous circuits are mapped onto synchronous FPGAs while maintaining their clockless nature and benefits [12]–[16]. In this context, ensuring timing integrity in the final FPGA implementation is a major challenge. Most design styles of practical relevance rely on timing assumptions in protocols, logic, or data transmission, which, if invalid, can cause failures. Fulfilling these assumptions on FPGAs is harder than on ASICs due to two main factors: the predefined hardware substrate and the large unpredictability of wire delays, which makes it difficult to close timing without conservative margins.

A key timing constraint in asynchronous circuit design consists of isochronic forks, crucial for the correct operation of quasi-delay insensitive (QDI) circuits, commonly used in state-of-the-art NoCs for their timing robustness [17]. Ensuring this constraint on FPGA targets is challenging [18], [19], and finding a generalized approach for handling QDI assumptions remains an active research area [20].

Recently, a less conservative design style is gaining momentum for NoCs, both for ASICs [21] and for direct mapping on FPGA [12], [22], showing promising improvements in key cost metrics such as coding efficiency, area, power, and performance. These NoCs rely on a *bundled-data* encoding scheme, which revolves around a synchronous-style datapath consisting of bundles of single wires per data bit, along with associated req/ack handshaking [1].

However, their FPGA mapping becomes even more challenging, because for their correct implementation a single one-sided relative timing constraint (RTC) must be satisfied, that the request delay is always longer than worst case data trans-

mission. *The convergence process of relative timing constraints is crucial for both functional correctness as well as final NoC performance.* The minimum delay constraint must in fact be applied to locally optimized data path delays, which conflicts with the focus of state-of-the-art tools on min/max delay constraints with absolute timing only. Additionally, overdesigning the relative timing margin (RTM) causes unnecessary idleness, resulting in a loss of performance. The ability of commercial CAD tools to predictably enforce specified RTMs has never been explored in depth so far.

Limited work has been done on synthesizing bundled-data circuits on commercial FPGAs so far. Existing approaches like [23], [24] generally focus on correctness only while ignoring performance optimization, require specialized tools and/or fail to support modern lightweight asynchronous pipelines.

The first systematic methodology for efficiently mapping bundled-data asynchronous designs on modern FPGAs using commercial tool suites was proposed by [12]. It achieves convergence of RTCs by slowing down the offending control signals through the insertion of localized delay in the RTL model, keeping the rest of the design unmodified. These delays are composed of buffer LUTs, and picked based on the differences by which the constraints are not satisfied.

This paper argues that the *brute-force* approach to relative timing convergence of the state-of-the-art methodology is too coarse-grained for accurate delay tuning and too broad in scope for fast and efficient optimization, since it requires rerunning the entire synthesis and implementation flow at each design iteration. As a result, it may largely overdesign relative timing margins, resulting in highly suboptimal NoC latency and throughput.

The main goal of this paper is to improve design technology for direct mapping of bundled-data NoCs on FPGA guided by an original focus on the predictable convergence of relative timing through a mainstream FPGA tool suite.

In order to gain a tight control over RTMs, the proposed methodology narrows down the optimization scope at each design iteration while still meeting the minimum delay targets, even with the simultaneous convergence of multiple constraints. The methodology consists of selectively unrouting violating request nets at each design iteration and of rerouting them based on dynamically-adjusted minimum delay constraints. The latter are fine-tuned to cope with the non-monotonic behavior of the heuristics used by commercial tools and to guide them effectively to convergence.

Should such post-routing optimizations fail to provide a valid routing solution meeting RTCs, a hierarchical tool flow is envisioned that progressively widens the optimization scope of each iteration (i.e., by rerunning only global routing first, or even the entire place&route) at the cost of computational complexity. LUT insertion becomes only the extreme course of action in case all other approaches fail.

Our experimental results indicate that selective net rerouting yields relative timing margins that are on average 43% lower than LUT Insertion while always fulfilling the target. As a result, implemented switches have roughly 30% better latencies and throughput, while saving 25% on occupied LUTs. In direct comparison with a 32-bit synchronous switch, on an Artix-7 FPGA asynchronous switch latencies are 40% lower and energy per packet is up to 75% lower, for matched throughput.

II. Background

A. Bundled-Data NoCs

The generic architecture of a 5x5 bundled-data NoC switch is reported in Fig. 1, including an expanded view of an Input Port Module (IPM) and of an Output Port Module (OPM).

In order to guarantee high-throughput operation, the architecture generally revolves around an asynchronous pipeline [1]. Similarly to single-cycle synchronous NoCs, the switch in Fig. 1 implements an input pipeline stage, decoupling the cycle time of the upstream link from that of the switch, and an output pipeline stage, decoupling the cycle time of the switch from that of the downstream link or output buffer (depending on the architecture configuration at hand).

Most bundled-data NoCs use Mousetrap pipelines [5], [21], [25], [26] or variations thereof [3], [27] because of their lightweight design and low-latency operation [28]. Mousetrap employs a two-phase protocol and single-rail bundled-data encoding (Fig.2a), offering similar coding efficiency to synchronous datapaths while being compatible with standard-cell design. Data flows elastically through the pipeline, coordinated by a "capture-pass" handshaking protocol (Fig.2b). Registers are typically transparent, closing only to protect data as it enters a stage and reopening when it moves to the next stage. Mousetrap's simple control circuits and data registers result in minimal area and delay overhead. Given its benefits and wide usage, this paper selects Mousetrap as the reference asynchronous pipeline.

In a NoC switch, packets are generally split into multiple words fitting the link bitwidth, called *flits*. When the head flit is captured into the input Mousetrap stage (Fig. 1), a *Packet Route Selector (PRS)* reads in header information and selects the productive output port towards the destination by driving signals *RSi* accordingly.

A *Request Generator (RG)* block is placed between the PRS and the OPM with the twofold goal of generating and keeping an allocation request (PPE_OPMi) for the target OPM and, for 2-phase communication protocols, adapting the polarity of the incoming request signal (ReqIn) with that of the OPM request signal (Req_OPMi).

OPMs arbitrate between multiple incoming requests trying to access the associated output channel. A key OPM component is the *N-way arbiter* [29], mediating between competing input requests in continuous time, without any reference clock.

Performance-driven asynchronous designs aiming for fast packet processing can encounter timing robustness issues. To address this, robust switch designs may include a grant *Masking Stage*, which quickly gates the arbiter output once the last flit of a packet is captured in the output Mousetrap. This prevents new input data from the same IPM from propagating through the OPM before it is reset for the next transaction.

The filtered grant signals concurrently select the correct data input of the *Crossbar Multiplexer* and enable only the winning request to propagate through the *Request Selection* block to the output Mousetrap stage, which then processes a new valid packet.

The output Mousetrap itself, or a separate dedicated circuit, then acknowledges the successful data capture into its latch-based registers (signals *Ack_to_OPM*), signaling completion of flit processing to the IPM through the *Ack Generator* block, which toggles a transition on the *AckX* signal regardless of the activated output and of its delay.

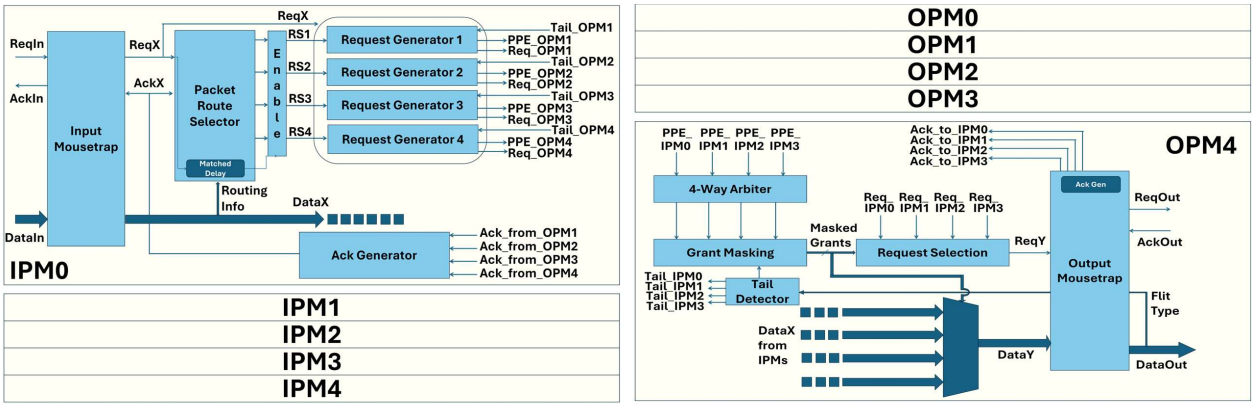


Fig. 1: Top level concept architecture of a generic 5x5 bundled-data switch.

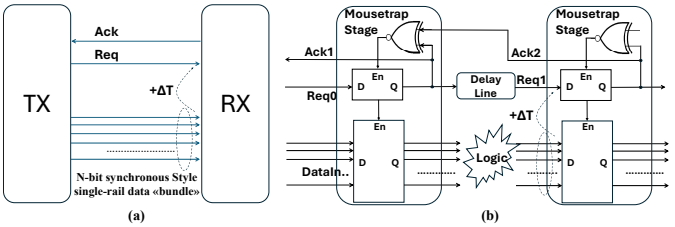


Fig. 2: (a) *Bundled-data* encoding. (b) *Mousetrap* pipeline.

B. Timing Constraints

A timing optimization flow for bundled-data NoCs has to deal with two kinds of timing constraints: those for *performance optimization* and those for *correct functionality*.

For optimizing the delay of the timing paths in the design, techniques like setting maximum delay constraints [12], [21] or defining virtual clocks [30] are used. Care should be devoted not to overload the synthesis engine, leading to sub-optimal results in complex designs.

Constraints for correct operation of bundled-data NoCs, also known as *bundling constraints*, instead consist of one-sided relative timing constraints (RTCs). In fact, the wires of each data channel must always be valid and stable before the corresponding request is observed at the receiver side (see the $+\Delta T$ relative delay in Fig. 2). For this purpose, the request delay must be always longer than the worst case data transmission delay. This bundling constraint is typically met by inserting a small matched delay on the control (i.e., request) line, when needed. However, there is no consensus on the most efficient way to do this through mainstream industrial CAD tools, especially for FPGA prototyping, where limited progress has been made so far.

Bundling constraints are typically applied in the following sections of the generic architecture shown in Fig. 1:

- At the input Mousetrap stage, the incoming *ReqIn* must be slower than the worst-case data bit delay from the incoming link.
- At the output Mousetrap stage, the incoming *ReqY* must be slower than the worst-case delay of the data path.
- In the switch control logic, a matched delay line typically runs in parallel to routing computation in the PRS, so to enforce hazard-free operation (see IPM details in Fig. 1).

Several difficulties make the enforcement of RTCs through commercial CAD tools complex and/or inefficient, especially:

- Control line delays should exceed datapath delays without overdesigning the relative timing margin (RTM) to avoid performance degradation. One solution is to enforce both min and max delay constraints on the control signal [21], but this can create contradictory constraints, making it hard for the synthesis tool to find a valid solution. Prioritizing the minimum constraint can alleviate this, but renders the maximum constraint ineffective. Instead of overloading the tool, this paper explores methods to insert matched delays on control wires with predictable outcomes on FPGAs.
- Unlike synchronous timing, RTCs are localized with no global constraint, enabling unbalanced stages in the NoC to interact with their own matched delays. This raises the challenge of simultaneously converging multiple bundling constraints without overloading the synthesis engine. This paper focuses on small, local optimizations to achieve computational affordability and effectiveness while ensuring constraints are met.
- Current commercial CAD tools target min/max delay constraints with absolute timing only. This makes it challenging to concurrently optimize the data path for performance and preserve the timing relationship between data channels and associated request signals for correct functionality. A common workaround is iterative synthesis: optimizing data channels, extracting delays, and tuning control signal delays. This is also the approach of this paper.

Other constraints for functional correctness are less critical, as they are usually met by typical gate and wire delays or addressed through existing design methodologies and techniques. These include asynchronous controller RTCs [31], [32], setup/hold time and data overrun constraints [28], [31], safe reset of components before reuse [5], delay-insensitive operation of asynchronous arbiters [29] and synthesis of asynchronous special cells like mutexes or C-elements [12].

While we account for these issues in our synthesis flow, they won't be the focus of the paper, as they are well-covered by existing literature. Instead, the focus will be on *protocol-level (i.e., bundling) RTCs*, which are crucial for both NoC performance and correct functionality. While some empirical work exists for ASIC bundled-data NoC synthesis [21], these methods do not directly apply to FPGA design due to differences in CAD tools, heuristics, and hardware.

III. Synthesis and Implementation Tool Flows

This paper focuses on a 2-step design methodology where a performance-optimized baseline implementation of a bundled-

data NoC switch is synthesized first, followed by an iterative timing optimization methodology of relative timing constraints. This way, the delay inserted in control signals can match locally-optimized datapath delays. The first step is discussed in Section III-A, while the second step is discussed in Section III-B (state-of-the-art approach) and in Section III-C (original paper contribution).

A. Performance-optimized baseline implementation

Methods for optimizing the performance of timing paths in the asynchronous design are well-established, and as such, they have been applied consistently across all the synthesis and implementation flows discussed in this paper (see the upper part of Figs. 3, 4 and 5, which is common).

The RTL model of a bundled-data NoC switch is used as the design entry. The HDL model then undergoes design synthesis and is constrained for performance. This optimization is guided by *set_max_delay* constraints, which ensure that signal propagation between design elements occurs within a specified maximum time. Since this step is not the primary focus of the paper, this work uses a fast methodology that applies an aggressive maximum delay constraint to all critical timing paths in the design and accepts the inferred delays by the tool as satisfactory. While this procedure might in some cases return suboptimal results when the heuristic spends excessive effort to reach an unrealistic target, in practice it works well for the experimental campaign of this paper, which documents largely superior performance over synchronous design (see Section IV-C). A more meticulous yet time-consuming methodology [12] could be used for performance fine-tuning, where relaxed maximum delay constraints are progressively made tighter till performance stops increasing monotonically.

The design flow then proceeds with the standard placement and routing steps to generate an initial FPGA implementation. At this point, timing information is extracted to check if minimum delay constraints are met, specifically whether *request delays are longer than the associated datapath delays by a specified relative timing margin (RTM).*

B. Matched delay through LUT insertion

A state-of-the-art technique to achieve convergence of RTCs when implementing bundled-data NoCs on FPGAs consists of slowing down the offending control signals through the insertion of localized delays *in the RTL model*, keeping the rest of the design unmodified [12]. These delays are composed of buffer LUTs. A complete synthesis and implementation flow using this technique is reported in Fig. 3. After a performance-optimized baseline is implemented, a list *L* of violated bundling constraints is compiled. Each offending request signal is then considered for LUT insertion, which implies a direct modification of the RTL model and a new round of synthesis, placement and routing. The iterative methodology completes when all RTMs are met.

This methodology takes a brute-force approach to minimum delay matching that guarantees convergence of violating RTCs at some point as LUTs are inserted. However, it comes with serious concerns. First, it incurs large computational complexity since a complete synthesis and implementation flow may have to be reiterated several times. Second, due to the wide scope of the optimization, at each iteration a new implementation may in principle differ significantly from the previous one, thus

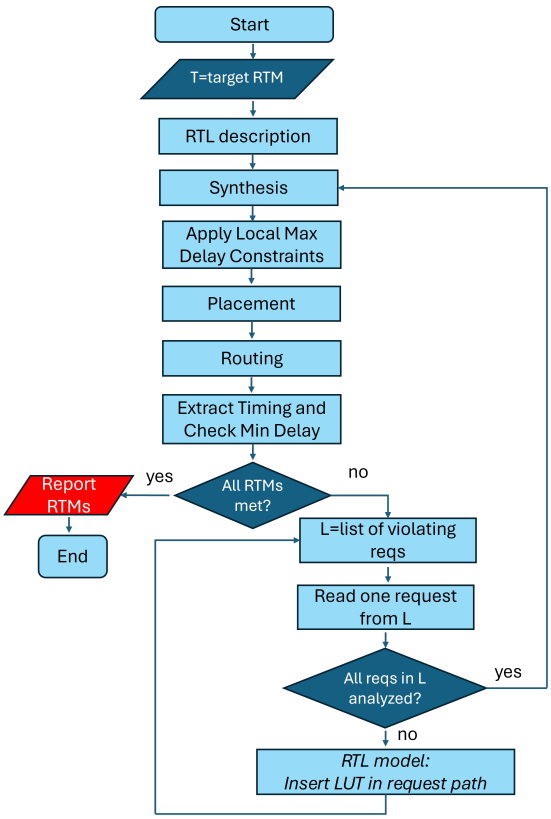


Fig. 3: State-of-the-art flow for LUT insertion.

challenging monotonic convergence. Third, LUT insertion consumes FPGA resources, especially in the presence of multiple RTCs and for long datapath delays to be matched.

In order to tackle these challenges, we propose a new timing optimization methodology that narrows down the optimization scope and gains fine-grained control over RTCs through small adjustments, thereby aiming at fast and predictable timing convergence.

C. Matched delay through selective net rerouting

The proposed synthesis and implementation flow revolves around a fine-tuning method of control signal delays by means of localized post-routing optimizations. The core approach involves starting with a fully performance-optimized initial implementation, then selectively undoing the routing of the request signals that violate their localized bundling constraint, and finally rerouting them under the specified minimum delay constraint till convergence, all while leaving the rest of the implementation (i.e., RTL model, placement, routing) unaffected.

In a mainstream industrial tool suite for FPGA design like Vivado, this is achieved through the $route\_design$ command and its $-unroute$ and $-pin$ options, specifying the end point of the request signal as the pin to be unrouted. A set_min_delay constraint for the discarded net is then applied, that sums up a RTM to the target minimum delay. Finally, the $route\_design$ command is executed once again to selectively regenerate net routing while hopefully meeting the target minimum delay, while leaving all other placement and routing decisions unaffected.

This methodology is overly fast with respect to LUT insertion: routing the request paths is roughly 20 times faster than implementing the project with a new LUT. For this reason, we

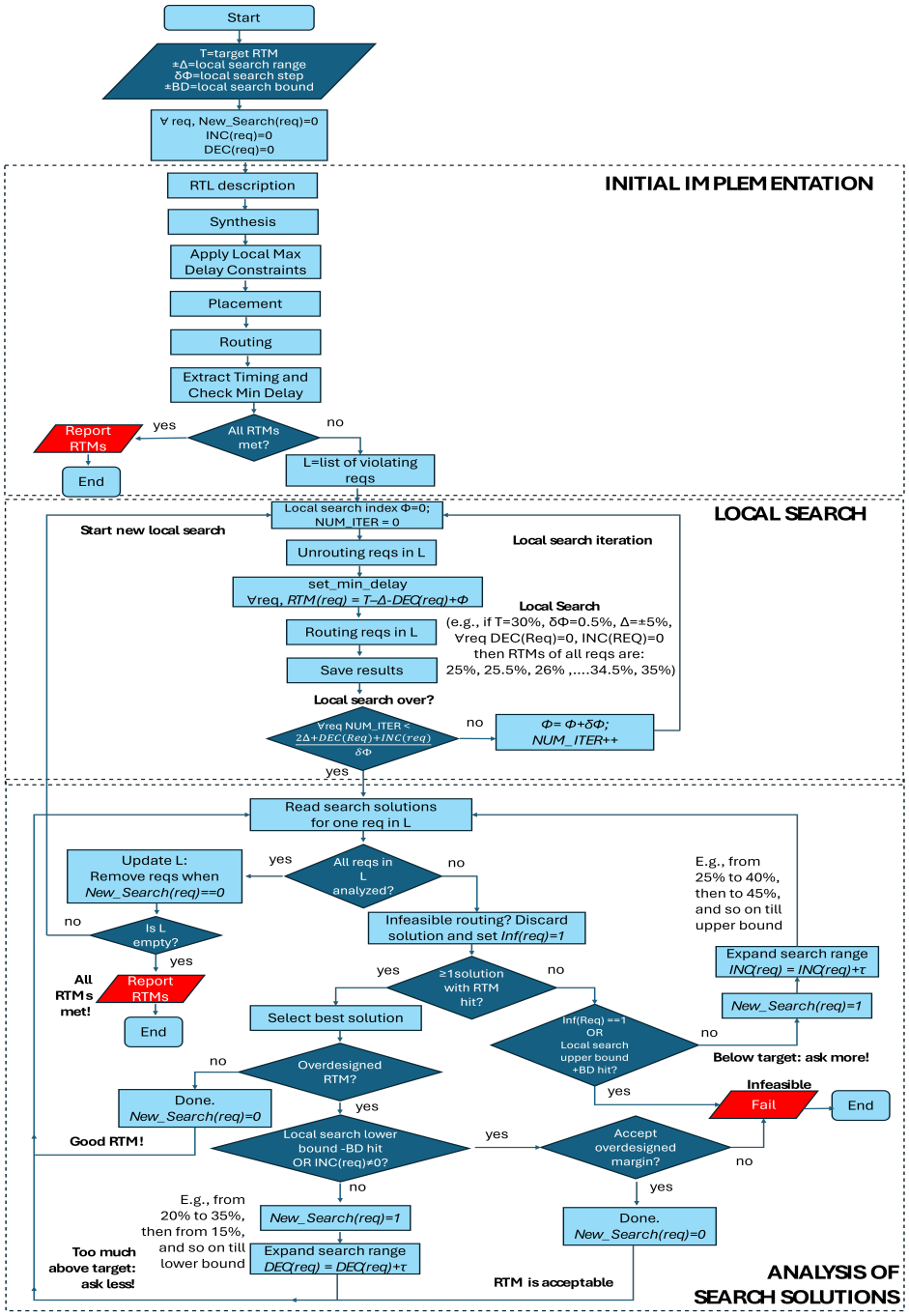


Fig. 4: Complete flow for selective request net rerouting.

reused a small part of the freed CPU time to run a local search around the target RTM. In fact, heuristics behind commercial tools may not always guarantee a monotonically improving solution (i.e., the search process may not always move one step toward the best possible solution) since their goal is to find a good enough solution in a reasonable time. A practical implication is that requesting a specific RTM value may not be the best way of achieving it, as slightly adjusting the value — either lower or higher — can sometimes result in a closer approximation of the target.

The complete flow is pictorially illustrated in Fig. 4. Initially, a performance-optimized baseline is achieved through the usual implementation steps. The remaining part of the flow is structured in two consecutive steps: *local search* and *search solution analysis*.

1) Local search: The violating request signals are recorded in a list L. Next, a predefined number FIN_ITER of fast post-routing optimizations is selectively performed on such violating nets, each time exploring an incremental RTM within a neighborhood of the target value (see the middle part of Fig. 4). The baseline number of iterations is given by:

$$FIN\_ITER = 2\Delta/\delta\Phi \tag{1}$$

where Δ is half of the local search range and $\delta\Phi$ is the search step. For instance, if a RTM of 30% is targeted with a search range of $\pm 5\%$, the local search will sweep margins from 25% to 35% in steps of 0.5% across 20 iterations. At each iteration, all request signals in L are selectively unrouted, constrained with increasing values of the RTM and finally rerouted. As a result, this procedure results in a *FIN_ITER-tuple* of obtained RTMs for each bundling constraint in L, corresponding to the explored RTM targets within the search range, spaced by the given search step.

In addition, we envision an outer optimization loop that, when needed, iterates the local search over asymmetrically extended ranges. For this purpose, the additional terms *INC(req)* and *DEC(req)* are added to the numerator of *FIN_ITER*, as hereafter explained.

2) Search solution analysis: At this stage, the *FIN_ITER-tuple* of solutions for each bundling constraint is analyzed in search of valid solutions fulfilling the target RTM. In this case, the constraint is removed from L, otherwise it is considered for a new round of local search with a modified range.

More specifically, if the tuple contains only solutions with insufficient minimum delays, then a new round of local search is scheduled with the search range extended upwards by an amount τ (see bottom-right branch of Fig. 4). For instance, stretching the upper extreme from 35% to 40% may drive the tool to provide at least the 30% target. The extension is recorded in the variable *INC(req)*, which adds up to the baseline number of search iterations to update the length of the new local search.[1] The search range is not extended indefinitely but only until an upper bound is hit. After that, if only violating and/or infeasible solutions are found for a constraint, the methodology fails.

A better scenario occurs when the initial *FIN_ITER-tuple* contains at least one routing option for the request signal under test that hits the RTM (see bottom-left branch of Fig. 4). In this case, the best solution is selected, which is the one that most closely approximates the target RTM from above (e.g., with a 30% target, 31% works better than 35%).

The proposed flow also checks for overdesigned RTMs, by comparing the synthesized RTM against a predefined upper bound (e.g., 50% if the target is 30%). Should the bound be exceeded, the flow schedules a new iteration of local search with the search range extended downwards by an amount τ. The extension is recorded in the variable *DEC(req)*, which adds up to the baseline number of search iterations to update the length of the new local search. When a lower bound of the exploration range is hit without any better solution, the designer may decide to either keep the overdesigned margin for a specific request, or to bail out.

Once all bundling constraints in L have been analyzed, the list is updated and a new local search is started in case it is not empty. Please notice that despite every surviving constraint in L may have different exploration ranges, the number of local search iterations is the same, thus leading to a common termination condition. The whole iterative procedure stops when there are no violating constraints left in L, and the final values of the matched request delays can be returned.

[1]In practice, it is not necessary to re-run net routing with the previously explored RTMs, since they could be simply recorded from the previous iteration of the local search.

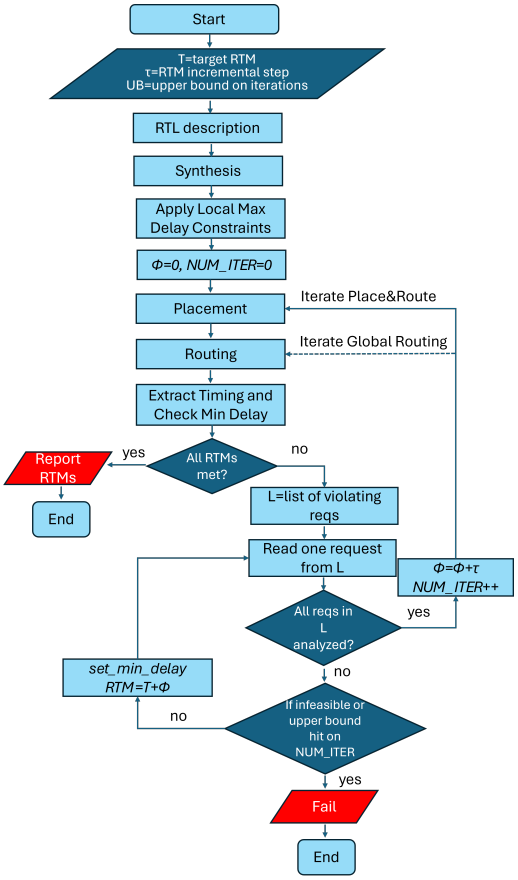


Fig. 5: Complete flow for place&route optimization.

D. Trading local refinement and computation efficiency for global optimization

LUT insertion and selective net rerouting can be viewed as two opposite approaches. The former ensures a comprehensive re-evaluation of the entire design, potentially offering more flexibility in optimizing RTCs across the whole design hierarchy. The latter takes a more targeted approach by selectively redoing only the last step of the implementation process. In principle, this approach might not fully capture broader optimization opportunities across earlier synthesis and implementation stages, and in some extreme conditions it might even fail to find legal routing solutions.

In order to cover these cases, we engineer two additional synthesis and implementation tool flows that bridge the conceptual gap between net unrouting/rerouting and LUT insertion and that have never been explored in the context of FPGAs so far. A common foundation of both methodologies consists of their use of set_min_delay constraints to insert matched delays on control signals, similar to net unrouting/rerouting. Their distinctive feature consists of spanning intermediate trade-offs between global re-optimization versus targeted, local refinement.

The first methodology reiterates only placement and routing in case bundling constraint violations persist (see Fig. 5). Such constraints are again recorded in a list, from which they are sequentially read and assigned a set_min_delay which initially sets the RTM equal to the target. After rerunning the implementation steps, the target RTM for violating request signals is increased by a predefined amount τ (e.g., from 30% to 35%). The iterative methodology completes when all RTCs

are met, or takes a failure exit in case place&route is infeasible or an upper bound on the number of iterations is hit without meeting the target RTC.

The scope of modification and the computational overhead can be further reduced by limiting the timing optimization loop to the global routing step only. In this case, the flow would be the same as in Fig. 5, but with the loop restarting from routing instead of placement (see dashed arrow).

E. Putting it alltogether: hierarchical flow

An interesting approach would be to view all the synthesis and implementation methodologies discussed so far as complementary rather than alternative. In practice, local net unrouting/rerouting could be seen as the reference methodology for quick fixes of relative timing without disrupting the entire design. Should this flow fail, it may trigger the activation of the separate flow that performs only global routing optimizations. However, since this higher-level flow can also fail due to the heuristic nature of the FPGA design tool, the entire place&route optimization flow could be called as a backup option. Finally, LUT insertion could be considered the final step when all other options have proven unsuccessful.

In our experimental campaign, selective net rerouting always meets the target RTMs, and the failure exit is never taken. However, the hierarchical flow may meet the needs of future NoC designs that could be significantly different from the one considered in this paper.

IV. Experimental Evaluation

The methodologies for RTCs illustrated in Section III are hereafter compared in terms of (i) design predictability (i.e., capability to meet a specified RTM on request paths while minimizing overdesign) and (ii) quality metrics of the NoC switches they synthesize, place and route. Finally, the best asynchronous switch is compared against a synchronous counterpart.

Without lack of generality, we consider a Xilinx Artix-7 (xc7a200tisbv484-1L) as the FPGA target, and the 2020.2 version of the Vivado design suite. Solutions under test are named as follows:

- *LUT Insertion*: the state-of-the-art approach reiterating design synthesis.
- *PlaceAndRoute*: each iteration implies a full reimplementation without design synthesis.
- *ReRoutingAll*: each iteration implies a full global routing without design synthesis and placement.
- *ReRoutingReq*: each iteration implies only the selective rerouting of violating request nets.

The goal is to assess whether widening the optimization scope (and complexity) yields better predictability and quality metrics. Without lack of generality, the state-of-the-art bundled-data NoC switch in [21] has been used as a testbench for the flows under test, due to the compelling advantages it has proven over competing synchronous and asynchronous designs.

A. Design predictability

The RTL model of a 5x5 instance of the bundled-data switch has been refined into an actual FPGA implementation. For the RTCs discussed in Section II-B (there are 10 of them in a 5x5 switch, excluding links), an increasing RTM has been targeted, expressed as a percentage of the datapath delays to be matched.

All flows have been instructed to check for RTM violations of the same design under test in both the fast *and* the slow corner of the FPGA, which are the two default process corners provided by Vivado for timing analysis, due to their relevance for synchronous design. Obtained RTMs are illustrated in Fig. 6, and have been measured through static timing analysis. Horizontal lines indicate the target RTM, and the height of bars over them indicates the amount of constraint overdesign.

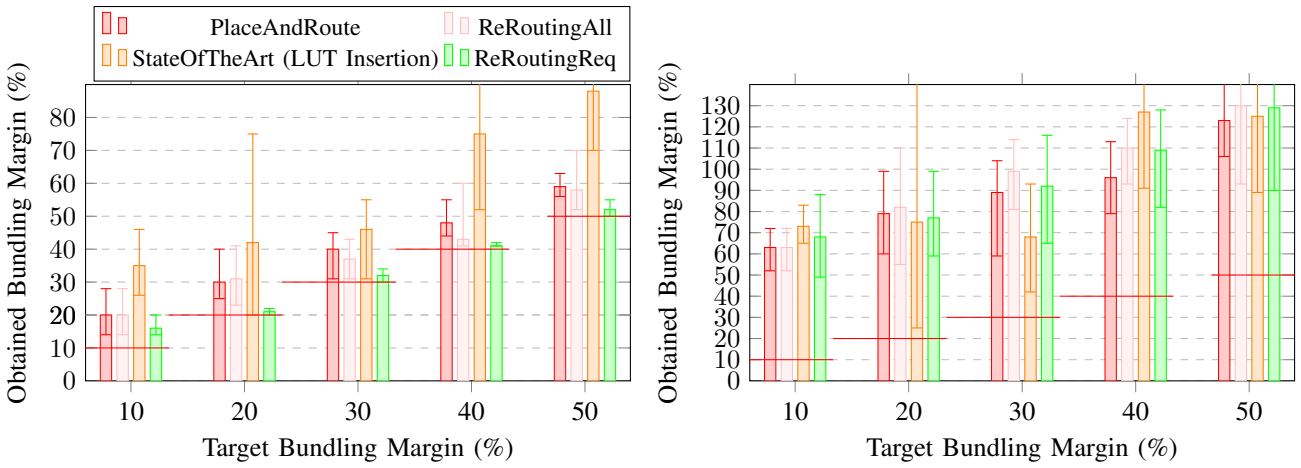
In the slow corner (Fig. 6a), *ReRoutingReq* is consistently the most predictable timing optimization flow. *LUT Insertion* overdesigns the bundling margin by 3.5x with a 10% target RTM (as opposed to 1.5x) and by 1.76x with a 50% one (as opposed to 1.04x). On average, the margins of *ReRoutingReq* are 43% lower than *LUT Insertion* while always fulfilling the target. These results prove that local post-routing optimizations are more predictable and suitable for fine-tuning of the design, while the state-of-the-art flow operates at a coarser granularity and with an unnecessarily wide optimization scope. The latter also comes with a side effect: when looking at min-max results for each bar in Fig. 6, it can be observed that *LUT Insertion* suffers from large variability. The ultimate outcome is an implemented switch with large performance variability across its I/O connections.

PlaceAndRoute and *ReRoutingAll* provide intermediate predictability results, confirming the intuition of Section III-E to use them as backup options in case the flow with less optimization effort and scope fails. Despite the conceptual interest of such a hierarchical flow for future applications, it is worth observing that in all the experiments of this paper no flow experienced any failure.

In the fast corner (Fig. 6b), the first observation is that all RTMs are largely overdesigned with all flows. This is because they easily meet the target RTM in the fast corner, but then keep adding delays to the request signals to meet the target in the slow corner too. Another observation is that all flows provide comparable predictability, unlike in the slow corner. This occurs because *LUT Insertion* ends up having more electronic devices on the request signals (as an effect of LUT insertion), while the other flows insert matched delays essentially through local vs. global routing or placement optimizations. As a result, request paths with *LUT Insertion* exhibit a larger variability when moving from the slow to the fast process corner, thus explaining the large reduction of the predictability gap between the competing flows.

Analyzing RTCs at slow and fast corners is necessary but insufficient to ensure functional correctness under all conditions. In fact, bundling constraints might be violated in intermediate conditions where timing assumptions for request and data signals diverge, such as when datapath or request delays shift between corners at different rates. As a post-processing step, we then analytically computed the correlation factor between requests and datapath delays that guarantees correct operation under all conditions. It turns out that when the target RTM ranges from 10 to 50%, the needed correlation factor for robust operation is relatively small and ranges from 20% to 35% (0 means no correlation, and 100% full correlation).

A more conservative approach consists of implementing the switches under the min-max delay assumption, that is, considering requests in the fast corner and data paths in the slow corner and meeting the target RTM in this pessimistic case. This approach guarantees correct behaviour under all PVT



(a) SLOW REQUEST / SLOW DATA

(b) FAST REQUEST / FAST DATA

Fig. 6: Obtained RTMs vs target ones in the (a) slow and (b) fast process corners.

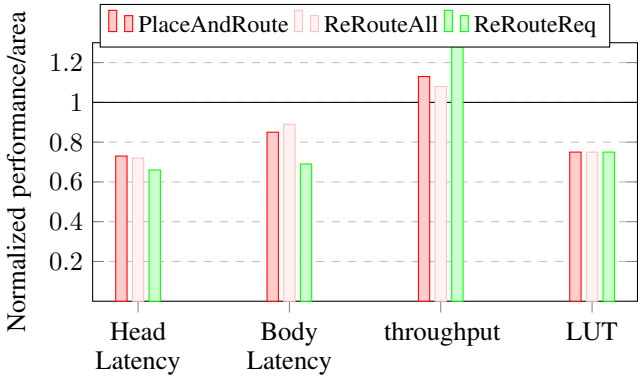


Fig. 7: Quality metrics of new flows normalized to LUT insertion.

combinations at the cost of an overdesigned RTM (i.e., large switch performance penalties) in most operating conditions.

The predictability of the flows under test is then compared when retargeting them for convergence under these timing assumptions. The target RTM was kept small (10%) not to add too much pessimism. Obtained RTMs indicate that while *ReRoutingReq* exceeds the target RTM by just 10%, the overdesign factor of the other methods ranges from 2.5x to 2.7x.

Overall, the choice of which timing assumptions to use to verify RTCs during switch implementation is a designer's decision. In any case, the proposed *ReRoutingReq* flow largely improves upon state-of-the-art in terms of the capability to gain a tight control over the inferred RTMs.

B. Switch quality metrics

The quality metrics of implemented designs by the flows under test are hereafter compared. For the sake of robustness, all implementations are verified against min-max timing assumptions (with 10% RTM), while presented quality metrics have been assessed in the slow corner, i.e., they reflect the worst achievable performance. Implemented designs include a 5x5 bundled-data switch and all 48 unidirectional links of a 4x4 2D mesh topology. For the latter, all switches have been placed in a grid throughout the whole FPGA, and spaced equally apart.

Figure 7 compares the head latency, body latency, throughput (obtained via post-implementation simulation) and FPGA resources of the 5x5 switches. Measured delays are averaged over all I/O communication paths of the switches. All quality metrics are normalized to those of *LUT Insertion*, which is considered as the baseline (see horizontal line).

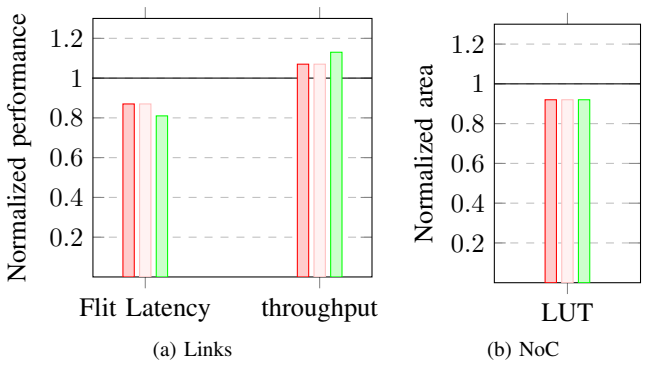
The three displayed flows exhibit roughly 30% reduction in head latency and 20% in body latency, uniquely indicating the performance benefits of inserting matched delays on control signals through minimum delay constraints and routing/placement optimizations instead of LUT insertion. *ReRoutingReq* achieves the best head latency (-35%) and body latency (-32%) due to the excellent design predictability characterized in Section IV-A. Similarly, the throughput gain of *ReRoutingReq* over *LUT Insertion* reaches 28%, with the other solutions effectively providing intermediate speedups (13% and 8%, respectively). Last but not least, since all the newly-proposed flows avoid LUT insertion, they cut down on the utilization of LUTs by 25%. In contrast, the number of FFs does not change, since it is design-specific, hence it is not showed.

Figure 8a shows average performance for all the 48 links of the NoC. Performance improvements with LUT-less delay lines reduce due to the higher effort to optimize wiring in an FPGA, and across long distances. Moreover, the large number of constraints handled simultaneously in parallel burdens the CAD tool (48 constraints vs. 10 for the switches). Nonetheless, *ReRoutingReq* still saves roughly 20% latency and improves throughput by 13%.

Finally, Fig. 8b shows the FPGA resources for the whole NoC (including all links and switches). Results indicate that LUT insertion for RTC convergence has a non-marginal impact over FPGA resources: *ReRoutingReq*, *ReRoutingAll* and *PlaceAndRoute* save roughly 10% in terms of overall LUT utilization.

C. Comparison with synchronous design

The asynchronous switch implemented with the proposed *ReRoutingReq* design flow has been compared with an open-source synchronous design from the ESP research platform for heterogeneous SoC design [33]. The comparison is between homogeneous architectures with minimum complexity and matched configurations: 5x5 ports, minimum input and output buffering for full throughput operation, xy routing algorithm. The synch. switch was synthesized for maximum performance,



(a) Links (b) NoC

Fig. 8: (a) Performance of NoC links and (b) resource utilization of a 4x4 2D mesh NoC, normalized to LUT insertion.

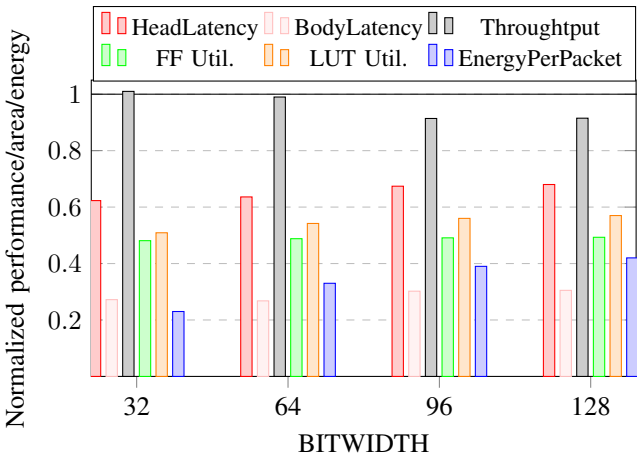


Fig. 9: Async. switch quality metrics, normalized to synchronous design

achieving a clock frequency of 130 MHz at 32 bits, which goes down to 125 MHz at 128 bits.

Performance, area and energy results are reported in Fig. 9 as a function of the bitwidth and normalized to those of the synchronous design. Latency includes the delay of traversing a switch and an ideal downstream link. Synchronous switches add latency in full clock cycles, while asynchronous switches adapt to actual link delays, in this case limited to propagation delay within the output Mousetrap. As a result, *ReRoutingReq* reduces head latency by 38% and body flit latency by up to 73%. The better body flit performance stems from the ability of asynchronous design to optimize flit-level performance, activating different timing paths depending on the flit type, unlike traditional clocked designs. Performance benefits over synchronous design are maintained when scaling the bitwidth.

Asynchronous design faces throughput challenges because of the time-consuming two-way req-ack handshaking protocol replacing the clock signal. Nonetheless, *ReRoutingReq* yields a matched throughput for 32-bit realizations. As the bitwidth increases, the criticality of the async. design becomes apparent, but the throughput drop is limited to 2% with 64 bits and to less than 10% with 96 and 128 bits.

Where instead a bundled-data async. design excels is at minimizing resources and energy consumption. With 32 bits, 52% less FFs and 50% less LUTs are used. With 128 bits, resource savings still amount to 51% and 43% for FFs and LUTs, respectively. Dynamic energy-per-packet was measured by backannotating the switching activity for the conflict-free transmission of 5-flit packets from all input ports. Again, the asynchronous switch turns out to be the most efficient solution, with energy savings that are as large as 75% with 32 bits and 58% with 128 bits.

This large efficiency gap is due to the lack of clock signals, the on-demand activation and the use of latch-based buffers, but is also amplified by the 2-slot buffers that the synchronous switch needs for full throughput operation, unlike the single-stage Mousetraps of the asynchronous realization.[2]

V. Conclusions

The main challenge in synthesizing modern bundled-data asynchronous NoCs is the efficient and predictable convergence of relative timing constraints. For FPGAs, this is typically achieved through LUT insertion on control signals. This paper proposes an alternative methodology that ensures tight control over these constraints while optimizing performance via local post-routing optimizations. The approach enforces minimum delay constraints on violating control nets before selectively rerouting them. Backup variants progressively widen the optimization scope without altering the original HDL model. This work advances FPGA implementations of bundled-data NoCs using commercial tools, improving performance, resource use, and power compared to synchronous designs.

[2]For full throughput operation, a synchronous flow control protocol requires a number of buffer slots equal to the round trip latency [34]. For asynchronous design, the flow control is built-in through the req/ack handshaking.

VI. Acknowledgments

This work was supported by the European Union through the TWIN-RELECT project under GA 101160314 and funded by UK Research and Innovation (UKRI) under the UK government's Horizon Europe funding Guarantee [grant number 10116095]. Research was also funded by the Italian Ministry of University and Research through PNRR - M4C2 - Investimento 1.3 (Decreto Direttoriale MUR n. 341 del 15/03/2022), Partenariato Esteso PE00000013 - "FAIR - Future Artificial Intelligence Research" - Spoke 8 "Pervasive AI", backed by the European Union under the NextGeneration EU programme".

References

[1] S. M. Nowick and M. Singh, "Asynchronous design—part 1: Overview and recent advances," *IEEE Design & Test*, vol. 32, no. 3, pp. 5–18, 2015.

[2] e. a. P. Vivet, "Intact: A 96-core processor with six chiplets 3d-stacked on an active interposer with distributed interconnects and integrated power management," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 1, pp. 79–97, 2021.

[3] M. Davies et al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[4] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[5] Z. Su, S. Ramini, D. Coffen Marcolin, A. Veronesi, M. Krstic, G. Indiveri, D. Bertozzi, and S. M. Nowick, "An ultra-low cost and multicast-enabled asynchronous noc for neuromorphic edge computing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 14, no. 3, pp. 409–424, 2024.

[6] O. R. et al., "A smart event-based vision sensor with a 320k neuron convolutional neuronal network processing pipeline," *ASYNC 2023*, 2023.

[7] Y. Y. et al., "Comparing loihi with a spinnaker 2 prototype on low-latency keyword spotting and adaptive robotic control," *Neuromorph. Comput. Eng.*, vol. 1, no. 1, 2023.

[8] J. V. Manoranjan and K. S. Stevens, "An a-fpga architecture for relative timing based asynchronous designs," in *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, 2014, pp. 1–6.

[9] H. S. Low, D. Shang, F. Xia, and A. Yakovlev, "Asynchronously assisted fpga for variability," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, 2014, pp. 1–4.

[10] R. Dashkin and R. Manohar, "General approach to asynchronous circuits simulation using synchronous fpgas," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 10, pp. 3452–3465, 2022.

[11] R. Dashkin and R.Manohar, "Mixed-level emulation of asynchronous circuits on synchronous fpgas," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2024.

[12] K. Bhardwaj, P. Mantovani, L. P. Carloni, and S. M. Nowick, "Towards a complete methodology for synthesizing bundled-data asynchronous circuits on fpgas," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2019, pp. 1–6.

[13] Q. T. Ho, J.-B. Rigaud, L. Fesquet, M. Renaudin, and R. Rolland, "Implementing asynchronous circuits on lut based fpgas," in *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, M. Glesner, P. Zipf, and M. Renovell, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 36–46.

[14] C. Pham-Quoc and A.-V. Dinh-Duc, "Hazard-free muller gates for implementing asynchronous circuits on xilinx fpga," in *2010 Fifth IEEE International Symposium on Electronic Design, Test Applications*, 2010, pp. 289–292.

[15] A. Mardari, Z. Jelčicová, and J. Sparsø, "Design and fpga-implementation of asynchronous circuits using two-phase handshaking," in *2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2019, pp. 9–18.

[16] Y.-F. E. Chang, R.-Y. Huang, and J.-H. R. Jiang, "Effective fpga resource utilization for quasi delay insensitive implementation of asynchronous circuits," in *2019 25th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2019, pp. 19–26.

[17] P. V. et al., "A $4 \times 4 \times 2$ homogeneous scalable 3d network-on-chip circuit with 326 mflit/s 0.66 pj/b robust and fault tolerant asynchronous 3d links," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 33–49, 2017.

[18] X. Wang, T. Ahonen, and J. Nurmi, "Prototyping a globally asynchronous locally synchronous network-on-chip on a conventional fpga device using synchronous design tools," in *2006 International Conference on Field Programmable Logic and Applications*, 2006, pp. 1–6.

[19] J. Quartana, S. Renane, A. Baixas, L. Fesquet, and M. Renaudin, "Gals systems prototyping using multiclock fpgas and asynchronous network-on-chips," in *International Conference on Field Programmable Logic and Applications, 2005.*, 2005, pp. 299–304.

[20] G. Mao, A. Yakovlev, F. Xia, S. Yu, and R. Shafik, "Automated mapping of asynchronous circuits on fpga under timing constraints," in *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 104–109.

[21] D. Bertozzi, G. Miorandi, A. Ghiribaldi, W. Burleson, G. Sadowski, K. Bhardwaj, W. Jiang, and S. M. Nowick, "Cost-effective and flexible asynchronous interconnect technology for gals systems," *IEEE Micro*, vol. 41, no. 1, pp. 69–81, 2021.

[22] H. Katabami, H. Saito, and T. Yoneda, "Design of a gals-noc using soft-cores on fpgas," in *2013 IEEE 7th International Symposium on Embedded Multicore Socs*, 2013, pp. 31–36.

[23] K. Takizawa, S. Hosaka, and H. Saito, "A design support tool set for asynchronous circuits with bundled-data implementation on fpgas," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, 2014, pp. 1–4.

[24] J. et al., "From click based asynchronous design to xilinx fpga," in *2018 Int. Symposium on Asynchronous Circuits and Systems*, 2018.

[25] M. Imai, T. Van Chu, K. Kise, and T. Yoneda, "The synchronous vs. asynchronous noc routers: an apple-to-apple comparison between synchronous and transition signaling asynchronous designs," in *2016 Tenth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, 2016, pp. 1–8.

[26] E. Kasapaki, M. Schoeberl, R. B. Sørensen, C. Müller, K. Goossens, and J. Sparsø, "Argo: A real-time network-on-chip architecture with an efficient gals implementation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 2, pp. 479–492, 2016.

[27] D. Gebhardt, J. You, and K. S. Stevens, "Design of an energy-efficient asynchronous noc and its optimization tools for heterogeneous socs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 9, pp. 1387–1399, 2011.

[28] M. Singh and S. M. Nowick, "Mousetrap: High-speed transition-signaling asynchronous pipelines," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 6, pp. 684–698, 2007.

[29] G. Miorandi, D. Bertozzi, and S. M. Nowick, "Increasing impartiality and robustness in high-performance n-way asynchronous arbiters," in *2015 IEEE Int. Symposium on Asynchronous Circuits and Systems*, 2015, pp. 108–115.

[30] N. Andrikos, L. Lavagno, D. Pandini, and C. P. Sotiriou, "A fully-automated desynchronization flow for synchronous circuits," in *2007 44th ACM/IEEE Design Automation Conference*, 2007, pp. 982–985.

[31] G. Gimenez, A. Cherkaoui, G. Cogniard, and L. Fesquet, "Static timing analysis of asynchronous bundled-data circuits," in *2018 24th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2018, pp. 110–118.

[32] M. Prakash and P. Beerel, "Static timing analysis of template-based asynchronous circuits," Patent US8 972 915, march, 2015, filed on: 2015-03-03.

[33] P. Mantovani, D. Giri, G. Di Guglielmo, L. Piccolboni, J. Zuckerman, E. G. Cota, M. Petracca, C. Pilato, and L. P. Carloni, "Agile soc development with open esp," in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3400302.3415753

[34] A. Pullini, F. Angiolini, D. Bertozzi, and L. Benini, "Fault tolerance overhead in network-on-chip flow control schemes," in *2005 18th Symposium on Integrated Circuits and Systems Design*, 2005, pp. 224–229.