Abstract

Asynchronous bundled-data (BD) circuits represent an important design paradigm for achieving modular, energy-efficient, and timing-robust computation in digital systems. Unlike conventional synchronous circuits, which rely on a globally distributed clock signal to coordinate operations, asynchronous circuits operate through local request–acknowledge handshaking between connected stages. This self-timed behavior allows each stage to run at its own data-dependent pace, eliminating global clock skew, reducing dynamic power associated with clock distribution networks, and improving resilience to variations in process, voltage, and temperature. The two-phase BD protocol in particular maintains a single-rail data path similar to synchronous architectures while enforcing correctness through the bundling constraint, which requires that the control path must always be slower than the corresponding data path. This balance between efficiency and practical implementability has made the BD protocol especially appealing for asynchronous processors and pipelines. However, despite their conceptual advantages, asynchronous BD circuits have historically remained difficult to implement on commercial FPGA platforms due to the clock-centric assumptions embedded within modern Electronic Design Automation (EDA) tools, especially in static timing analysis (STA) and delay placement.

FPGA architectures impose unique challenges not encountered in ASIC-oriented asynchronous design methodologies. In an FPGA, delays are realized through the discrete timing characteristics of lookup tables (LUTs), flip-flops, carry chains, and routing multiplexers. These components provide only coarse, unpredictable, and placement-dependent delay resolutions, which complicate the precise alignment required for correct BD operation. Conventional FPGA STA engines assume the presence of clock signals as the primary timing reference and lack direct support for expressing handshake causality, minimum-delay behavior, or relative timing constraints between data path and control-path components. As a result, the timing correctness of asynchronous circuits depends heavily on manually constructed constraints, hand-tuned delay elements, and repeated place-and-route iterations. Slight changes in routing decisions can shift critical delays enough to violate handshake sequencing or bundling requirements, further complicating convergence. Consequently, asynchronous circuit designers have traditionally relied on iterative timing-closure methodologies that demand substantial expertise, manual effort, and extensive experimentation. These conventional approaches, including Adaptive Delay Matching (ADM), Propagation Timing Constraint (PTC), and Backward Delay Propagation Constraint (BDPC), attempt to bridge the gap between handshake semantics and synchronous STA tools but each suffers from inherent limitations such as unsupported minimum-delay constraints, incorrect assumptions regarding phase propagation, inability to properly model loop behavior, and dependence on post-layout Engineering Change Order (ECO) operations that jeopardize timing reproducibility.

To address these critical challenges and enable the practical implementation of asynchronous BD circuits on FPGA platforms without requiring modifications to commercial tools, this dissertation proposes a unified design framework that integrates two complementary methodologies: the Generated Clock Propagation (GCP) method for timing analysis and the Morphological Delay-Placement Constraint (MDPC) method for delay insertion. Together, these methods create a semi-automated, deterministic, and FPGA-compatible flow that achieves timing closure in a single synthesis pass while maintaining high precision, scalability, and resource efficiency. The central motivation behind this framework is to reconcile the self-timed behavior of two-phase BD circuits with the synchronous assumptions of FPGA STA engines by constructing a mapping that preserves handshake semantics, enables robust slack computation, and ensures placement-aware delay balancing.

The GCP method in this thesis reframes handshake timing behavior into a form that can be interpreted by existing STA engines without modifying tool internals. Instead of treating handshake signals as arbitrary, tool-agnostic transitions, GCP interprets each request event as a virtual clock domain. By constructing a network of generated clocks, the timing relationship between handshake

events can be expressed using standard commands such as create_clock, create_generated_clock, set_clock_groups, and set_false_path. Through this formulation, GCP allows the STA engine to evaluate setup and hold constraints across complex asynchronous networks in exactly the same manner as synchronous multi-clock systems. The propagation of virtual clocks follows the causal structure of the handshake network, enabling the STA engine to derive accurate timing paths and slack values. An important aspect of GCP is its ability to identify and isolate architectural loops, such as those found in sequential-ring circuits or iterative feedback pipelines, as well as local combinational loops within handshake controllers. Without such loop isolation, STA engines tend to propagate generated clocks indefinitely, leading to infinite or unstable slack values. GCP strategically disables non-functional timing arcs, prunes loop-back paths, and stabilizes propagation to yield correct, finite timing information. Furthermore, the method correctly handles conditional behavior through selective activation of handshake branches, enabling the STA engine to analyze only one active path at a time in circuits involving MUX, DEMUX, fork, join, and passive controllers. This per-path analysis is particularly important in architectures employing multiple selection-delay channels, such as frequency-adaptive pipelines or multi-mode processors. By fully leveraging native STA commands, GCP achieves accurate and loop-safe timing analysis for asynchronous circuits, providing consistent results regardless of placement randomness or routing variations.

The second contribution, the MDPC method, addresses the complementary challenge of timing closure through physically aware delay insertion. Traditional asynchronous FPGA flows often rely on hand-crafted delay chains or heuristics that place delay elements arbitrarily near control-path logic. However, because FPGA placement heavily influences routing delays, manually inserted delay lines frequently fail to match data path timing precisely or result in unstable timing behavior across synthesis iterations. MDPC introduces a novel placement strategy rooted in morphological image analysis to determine optimal delay locations. The method begins by representing implemented circuit regions as geometric patterns on the FPGA grid. Using dilation, erosion, and subtraction operations, MDPC identifies the physical boundary between datapath logic and control-path logic, forming a candidate region where delay elements would have the greatest likelihood of producing predictable timing behavior. Once candidate regions are extracted, MDPC performs slack-guided refinement: for each candidate tile, a temporary delay element is inserted, and slack is extracted. The candidate producing a slack value closest to the target requirement is selected as the optimal insertion site. This iterative evaluation continues until the desired control delay is achieved. Because MDPC relies directly on timing information obtained after placement and routing, its choices reflect the true physical characteristics of the design, rather than inferred or estimated delay behavior. As a result, MDPC achieves deterministic timing closure without requiring repeated design iterations or post-layout manual editing. This single-pass property marks a substantial departure from previous methodologies and significantly improves both design efficiency and precision.

The unified GCP + MDPC framework forms a complete asynchronous FPGA design flow compatible with conventional FPGA toolchains. The flow begins with HDL design entry using standard two-phase handshake primitives such as Click and its phase-decoupled variants. After initial synthesis produces a technology-mapped netlist, GCP generates timing constraints that accurately reflect handshake dependencies, loop structures, and selection behaviors. Placement and routing proceed under these constraints, ensuring that timing relationships remain stable across implementation. MDPC then inserts delay elements within the physically meaningful boundary regions guided by slack feedback. Final STA confirms that all setup and hold requirements are satisfied, after which the standard toolchain generates the FPGA bitstream. Throughout this process, no modifications to the underlying tools are required, and the entire flow is reproducible across routing seeds and device families.

The effectiveness of the proposed framework is demonstrated through its application to five asynchronous RISC-V processor pipelines, including linear, selective, frequency-adaptive, reduced, and combined styles, each representing different architectural structures and handshake characteristics. These designs encompass a broad range of behavioral patterns, such as pure feed-forward pipelines, pipelines incorporating conditional selection-delay channels, adaptively controlled timing paths,

reduced-latency data paths, and hybrid architectures that integrate multiple asynchronous mechanisms. Experimental evaluation on Xilinx FPGA platforms using Coremark confirms that the GCP + MDPC flow achieves functional correctness across all implementations while substantially improving timing accuracy, performance, and energy efficiency compared with conventional iterative flows. Across the evaluated pipeline styles, processing speed improved by up to 13.50%, and energy consumption decreased by as much as 11.59%. Timing behavior remained stable, with slack deviations consistently within 1.45%. Furthermore, the flow significantly reduced hardware overhead, lowering LUT usage for delay elements by 3.76–5.56×. Collectively, these results highlight the practical advantages of the proposed methodology and establish its suitability for scalable and deterministic asynchronous processor design on FPGA platforms.

Beyond quantitative gains, the experimental evaluation also highlights the conceptual advantages of the proposed framework. By semi-automating both timing analysis and delay placement, the methodology eliminates the historically difficult and error-prone steps of manual constraint editing and delay tuning. Because delay insertion is guided by physically grounded geometric analysis, MDPC maintains timing robustness across routing or synthesis variations, significantly improving reproducibility. At the same time, the GCP method transforms asynchronous STA into a representation that mainstream FPGA tools can naturally interpret, enabling designers to analyze complex handshake pipelines without custom scripts or external analyzers. Building upon these capabilities, this dissertation establishes a comprehensive, deterministic, and FPGA-compatible design flow for asynchronous BD circuits. GCP resolves long-standing limitations in modeling loops, conditional paths, and multi-channel timing behavior, while MDPC complements it by automating delay insertion in a single implementation pass through geometry-aware extraction and slack-driven refinement. Together, these techniques bridge the conceptual gap between self-timed design principles and the synchronous assumptions embedded within commercial FPGA tools. Through extensive validation across multiple asynchronous RISC-V pipeline architectures, the combined framework demonstrates that self-timed processors can be implemented with consistent timing, reduced resource usage, improved performance, and enhanced energy behavior, all within unmodified FPGA toolchains, showing that the gap between asynchronous theory and FPGA practice can be substantially narrowed and positioning asynchronous computation for broader use in future low-power, high-efficiency, and adaptive digital systems.