6. CONCLUSIONS

6.1 Summary

NCL is fundamentally *incompatible* with commercial synchronous-targeted EDA tools due to the input-completeness and observability requirements. MTNCL, on the other hand, does not have these restrictions, making it more compatible with such tools. This dissertation presents the first design flow that transitions from standard synchronous behavioral RTL to highly optimized and reliable physical implementations of MTNCL circuits. The flow can be divided into two primary components: the synthesis flow and the novel timing constraints applied during placement and routing.

While this work is not the first to propose a synthesis flow for MTNCL circuits, existing flows have one or more notable drawbacks: requiring custom modifications to the source RTL, being limited to a specific HDL, lacking register retiming, supporting limited PPA enhancements, and structurally instantiating completion detection logic. Furthermore, none of the flows provide an acceptable approach to buffering *sleep* trees, and their evaluations are limited to gate-level netlists. In contrast, the synthesis flow developed through this work is subject to *none* of these limitations.

This work is the first to comprehensively document all four of MTNCL's race conditions. Two novel sets of timing constraints are developed; the first of which can be used to safeguard MTNCL circuits against these hazards. Timing closure with these constraints yields highly reliable, fully modular MTNCL circuits, enabling a team to compose an arbitrary configuration of subblocks with no reliability concerns, even at the circuit boundaries. A broad range of PVT corners can be employed, potentially covering near threshold and subthreshold operation while still guaranteeing reliably. The second set of timing constraints can be used to optimize MTNCL

circuits for high performance, low active energy per operation, low leakage power, or some target in between.

Three circuit types—64-bit adders, 32×32 Montgomery modular multipliers, and AES-256 cores—were used to evaluate the synthesis flow and timing constraints. Two versions of each circuit, *high performance* and *low power*, were implemented to provide a well-rounded evaluation. Overall, the RTL-to-GDS flow achieved higher performance and lower power consumption when compared with *structural* designs and those implemented without timing constraints. Only exceedingly simple *structural* designs, such as the RCA, were comparable. The PPA gains achieved through the application of the flow range from 140% higher throughput for *high-performance* circuits to the combination of 28.6% less active energy per operation, 42.9% less static power consumption, and 29.6% less area for circuits targeting *low-power* operation. Far more valuable than these PPA gains, however, is the reliability imparted to the designs from the timing constraints, making them the *only* viable choice. Although the fundamental differences between synchronous and MTNCL circuits, like *worst-case* vs. *average-case* performance, prevent existing commercial EDA tools from being an ideal solution to the design of MTNCL circuits, application of these tools in place of *structural* design without timing constraints is *imperative* to achieving high reliability, high performance, and low power.

The vast majority of existing literature on MTNCL is based upon *structurally* designed circuits. Additionally, as elaborated upon in the next section, many of the other existing MTNCL synthesis flows share a substantial issue, degrading the performance, power efficiency, and area of the resultant MTNCL circuits. For these reasons, previous works indicating that MTNCL circuits are slow, power inefficient, and/or large, sometimes in comparison to synchronous alternatives, may not reflect intrinsic drawbacks to MTNCL but rather shortcomings in how the

circuits were designed. Further work is necessary to more accurately measure and analyze these metrics for MTNCL circuits.

6.2 Pitfalls

This section discusses some of the pitfalls discovered during the development and optimization of the synthesis flow and timing constraints. The initial version of the synthesis flow utilized MTNCL cell libraries instead of Boolean libraries during single-rail synthesis. While synthesizing the *low-power* adder, it became apparent that Genus was looking for Boolean cells like half adders and full adders. Without these standard cells, Genus could not produce an RCA and would instead generate a larger and higher power adder. In short, Genus requires a standard set of Boolean cells to complete high-level datapath optimizations, such as selecting multiplier architectures.

The output of single-rail synthesis in the original flow also differed. After standard generic mapping, technology-specific mapping, and optimization, the logic equations describing the target circuit were written out in Verilog with *write_hdl -equation -generic*. Step four then directly implemented the dual-rail expansion transformation depicted in Figure 17 and Figure 18 using a Java program, which ran quickly and scaled well with increasing circuit size. However, Genus attempted to further optimize the circuit during the generic step of dual-rail synthesis, causing an *explosion* in the amount of circuit logic. The inflation was so severe that the original set of *optimized* circuits was consistently and noticeably slower, higher power, and larger than the *structural* alternatives.

To resolve this issue, the single-rail synthesis step was modified to instead write out the netlist using the Boolean gates. However, even when Genus is provided a gate-level input netlist during dual-rail synthesis, it will *unmap* the design during generic synthesis and trigger the same

issue. This likely stemmed from a feature known as technology translation, where an already synthesized netlist is mapped to a new library. To prevent this, the second Genus run was prohibited from unmapping the Boolean gates through setting the *dont_touch* attribute. Following the generic step, the *dont_touch* changes were rolled back, allowing Genus to flatten the Boolean gates into their dual-rail MTNCL equivalents and further optimize the already mapped circuit. Prohibiting Genus from unmapping the logic gates during the generic step of dual-rail synthesis was essential to achieving the high-quality *synthesized* and *optimized* circuits presented in this work.

Interestingly, this mapping issue seems specific to *arithmetic* circuits like the adders and Montgomery modular multipliers, while the QoR discrepancy for a logical circuit like the AES-256 core is minimal before and after the fix. This suggests that during dual-rail synthesis of arithmetic circuits, Genus is unaware that the provided netlists already contain high-quality, efficient component architectures from the single-rail synthesis step. When Genus is provided a set of Boolean equations or *allowed to unmap the circuit* at the beginning of dual-rail synthesis, it views the design as a complex logic circuit and attempts to optimize it. However, this *destroys* the pre-existing, highly efficient arithmetic implementation, leading to *severe* PPA degradation.

Testing with a wider range of circuits and alternative synthesis utilities like Design Compiler is needed to better understand this issue. Unfortunately, most of the pre-existing MTNCL synthesis flows provided unmapped netlists to dual-rail synthesis or allowed the utility to unmap the design. Thus, metrics in these works indicating that MTNCL circuits are slower, more energy inefficient, or larger than their synchronous counterparts may be partially attributable to this synthesis flow issue rather than inherent deficiencies in MTNCL itself. In fact, this issue was only uncovered in this work because of the comparison with structural equivalents. Further

investigation and eventual discovery of this issue only transpired in response to the original, *very* counterintuitive trend that the *structural* designs were far superior to the *synthesized* and *optimized* implementations.

6.3 Future Work

There are several avenues to extending the synthesis flow and timing constraints in future work. The current synthesis flow only directly supports fully combinational, linear pipelines. Synthesizing the AES-256 cores used in this work required a non-trivial amount of manual work to transform the original design into two separate single-stage combinational designs for the key expansion and round logic. The flow can be improved to support sequential designs and, ideally, highly complex mixtures of both logic types.

Switching from Boolean standard cell libraries in single-rail synthesis to MTNCL cell libraries with different power and performance characteristics during dual-rail synthesis creates discontinuity, disrupting Genus's ability to optimize the target circuit. The ideal solution would be to construct a library of Boolean standard cells *out of* the MTNCL cells. Single-rail synthesis would have the cell types it is looking for (e.g., full adders) but the transition to dual-rail synthesis would be smooth given the same underlying cell power and performance characteristics. Targeting MTNCL's *average-case* behavior, the author recommends constructing dual-rail Boolean cells, with constituent logic for both RAIL1 and RAIL0. Following characterization, the delay, power, and noise arcs of the two rails can be averaged to get a set of arcs for the single-rail *image* cells that exemplify the *average behavior* of the eventual dual-rail versions. Selecting the ideal MTNCL threshold gates while creating this Boolean cell library is likely not straightforward, so the best approach might be to write behavioral descriptions of each

Boolean cell comprising the equations for both rails and then synthesizing the cells, allowing the synthesis utility to select the ideal threshold gate implementation.

This proposed improvement to the single-rail synthesis cell library might also apply to inverter cells. For comparison, if a single-rail signal in a Boolean circuit needs to be inverted, the introduction of an inverter carries a penalty to delay, power, and area. However, in dual-rail logic, the inversion of any signal is already available through Rail0 and thus carries *no penalty*. Consequently, if inverting a signal during single-rail synthesis assists with optimization, the utility ought to proceed with this transformation instead of doing a cost-benefit analysis against any aforementioned penalties. To this end, the author recommends evaluating the merits of a *free inverter*. It might be possible to modify the inverter's Liberty model and perhaps apply Genus's delay multiplier to effectively signal to Genus that the inverter has no power consumption and no propagation delay. The synthesis utility would then be free to use inverters wherever advantageous. When the output of single-rail synthesis is transformed into a functionally equivalent dual-rail circuit, the inverters no longer exist, resulting in the *same* zero delay and power overhead associated with the *free inverter*. However, rigorous analysis is necessary to ensure that the modified inverter does not introduce unintended negative consequences to the timing analysis and optimization of single-rail synthesis.

While the recommendations thus far have exclusively pertained to the synthesis of MTNCL circuits, there is also room for improvement during physical implementation. It became apparent while working to close timing on the presented circuits that the chosen timing constraints could be improved to facilitate timing closure and potentially improve PPA, without compromising the reliability and performance benefits of the original paths. A clear first choice for improvement would be the NULL completion timing constraints, *E* and *F*. The current constraints ensure every

single gate in the stage enters sleep mode before the end of the completion *AND tree* enters sleep mode, which is challenging to achieve in *high-performance* circuits. The primary issue with this approach is that it does not leverage the delay from the launch point at the output of the completion detection through the upstream completion detection and *sleep* tree as effectively as possible. Instead, this delay could be used as a *margin* between the min and max delay constraints, easing timing closure. A potential alternative might be to begin *E* and *F* at the *sleep* pins of the MTNCL gates in the current stage. Here, *E* would ensure that all the gates enter sleep before *F*. Conversely, *F* would comprise the *fastest* gate's transition to sleep mode, the current completion detection's request for DATA, the preceding completion detection's request for NULL, and the propagation of sleep through the tree until it arrives back at the launch point which deasserts the *sleep* of the *first* MTNCL gate in the stage. While this specific approach may or may not prove advantageous, it illustrates how the delay through the constituent components along the path can be utilized more effectively. More generally, a better set of timing constraints has the potential to further improve the PPA presented in this work.

The precise values chosen for the reliability-related constraints can have significant impact upon the active energy, leakage power, and area of the resultant circuit. It was observed that if either the set *E, F,* and *3* or the set *G, H,* and *4* are set too high, Innovus will add an excessive number of buffers to close hold timing. The same effect can occur if *C* and *D* are set too high or too low; therefore, iterative adjustments were required on the presented circuits to maintain a reasonable number of driver cells and achieve competitive PPA values. This was especially true for the RCA, where an increase in the number of drivers substantially degraded PPA given the small circuit size and *low-power* goal. Establishing a standard for how to choose these values or creating a more automated approach would be beneficial.

Finally, constraining and closing timing on structurally complex designs like the AES core is not simple. Additional paths, primarily handshaking related, must be identified and constrained to ensure reliability and achieve quality PPA. To address this challenge and ensure paths are not overlooked, the generation of timing constraints for a given MTNCL circuit should be automated. For the AES-256 cores, due to their more complex structure, timing closure was difficult, requiring many iterations of atomically changing various delay values to identify which changes needed to be rolled back in response to overall timing degradation. To ensure that none of the constraints conflicted with one another, a total of *four* separate modes were created in the MMMC configuration for these designs, a sharp deviation from the single mode used in the other circuits. Further investigation is needed to improve this process, employing automation where possible. Most of these issues might be solved by simply improving the way the constraints are specified as advocated for earlier in this section.