Commercial synthesis utilities require a basic set of cells before synthesis can be attempted, typically a flip-flop, inverter, and either an *AND* or *OR* gate. This requirement poses a challenge for the seventh step of the synthesis flow. Standard, monotonic dual-rail logic like MTNCL cannot contain negative unate cells like inverters. A subset of the inverters present in the combinational logic would output a '1' during NULL waves, and when the stage is taken out of sleep, these '1's would be interpreted as DATA, effectively creating a glitch that could corrupt the circuit's output. To the author's knowledge, there exists no method to fully prohibit Genus from using inverters in the target circuit. Any attempt to disable all inverter cells—e.g., using a *dont_use* attribute—results in an error indicating that the synthesis operation requires at least one inverter library cell. From a single-rail synchronous paradigm, this requirement is rational and presents no issues. In fact, almost all single-rail synchronous circuits will require negative unate cells to implement their underlying logic. Unfortunately, commercial synthesis utilities are developed for single-rail synchronous circuits, leaving no way to definitively avoid this issue without support from EDA companies.

Given that the input RTL to step seven is dual-rail and comprises exclusively positive unate logic, Genus *can* successfully synthesize the target circuit without any inverters. The key is to *bias* Genus's cell selection. The workaround employed in this dissertation involves making inverter cells appear expensive in regard to delay, power, area, or some combination of the three by modifying the Liberty model or employing Genus's cell delay and area multiplier attributes. Consequently, Genus will rationally avoid inserting inverters in pursuit of optimized PPA. A paramount requirement of this solution is disabling design partitioning using the *auto_partition* attribute. For sufficiently large circuits, Genus partitions the design into smaller components to enable parallel synthesis, thereby reducing overall runtime. Genus, viewing the dual-rail circuit

as single-rail, may partition the underlying logic equations into negative unate blocks which require inverters or other negative unate cells to correctly implement, despite their artificially inflated cost. After configuring these settings and proceeding through physical synthesis, step seven outputs a gate-level netlist.

Conducted separately from step seven to allow for greater optimization, step eight entails synthesis of the input completion component. If this subblock was included as an additional stage to step seven's synchronous pipeline, the clock period would far exceed its delay, forgoing optimization. Although separate, step eight follows the same approach and adheres to the same requirements as step seven.

3.2.4 Registration and Handshaking Connection

The nineth step of the flow divides the gate-level netlist produced in step seven into individual logic stages using a Java program. Subsequently, *sleep* pins are added to each MTNCL gate instance, and the associated *sleep* signal is added as a port for each stage using a simple BASH script. Step 11 constructs the final circuit by instantiating the individual logic stages, the input completion detection, and the registration while connecting the handshaking signals. This hierarchical design is verified using Questa in step 12, followed by flattening with Genus in step 13. Finally, the complete flattened design is logically verified one last time with Questa in step 14.

3.3 MTNCL Timing Constraints

As discussed in Section 2.7, all existing MTNCL synthesis flows end with a logically correct MTNCL netlist. While these netlists serve as a good proof of concept, their utility is limited. Physical implementation is required before any solid-state circuit can be fabricated and deployed. Further, in today's advanced process nodes, netlists produced from logical synthesis

are no longer reliable indicators of performance and power consumption. Demonstrated by the advent of physical synthesis, consideration of layout-level effects is crucial to accurately assessing these metrics. This work aims to go beyond an interesting proof of concept, enabling end users to create real MTNCL circuits by guiding the physical implementation process.

Therefore, the second major contribution of this work, in addition to aggregating and enhancing the existing knowledge on MTNCL's relative timing assumptions, is to provide a set of timing constraints that, when satisfied, ensure reliable operation of the resultant MTNCL circuit. This set of constraints enables the generation of truly modular MTNCL circuits, alleviating the need for timing analysis at the interfaces during integration. Furthermore, this section provides an additional set of constraints that guide physical implementation tools to optimize MTNCL circuits for high performance, low power, minimal area, or a balance among the three.

Before meaningful timing analysis can be attempted on MTNCL circuits, all combinational timing loops must be broken. This requirement is not unique to NCL and MTNCL as all asynchronous templates contain combinational loops, and commercial EDA tools lack support for this circuit construct [23, 34, 35]. As clearly indicated in Figure 22 via the red and blue paths, a combinational loop exists between each pair of adjacent stages. Further, the green path demonstrates that loops can also be constructed through three or more completion units. These handshaking loops always pass through completion detection units, so breaking the paths there is a natural choice. More precisely, this work chooses to break the loops using the NCL *TH22n* cell within each completion detection unit by disabling the timing arc from the *a* pin of the cell, connected to the MTNCL *AND tree,* to the output pin of the cell, *z*. In contrast, the arc from the *b* pin, and thus *Ki*, to the output pin is preserved. Breaking the loops in this manner can be

accomplished using the Synopsys Design Constraints (SDC) command provided in Figure 23 when supplemented with the appropriate regular expression for the cell library.

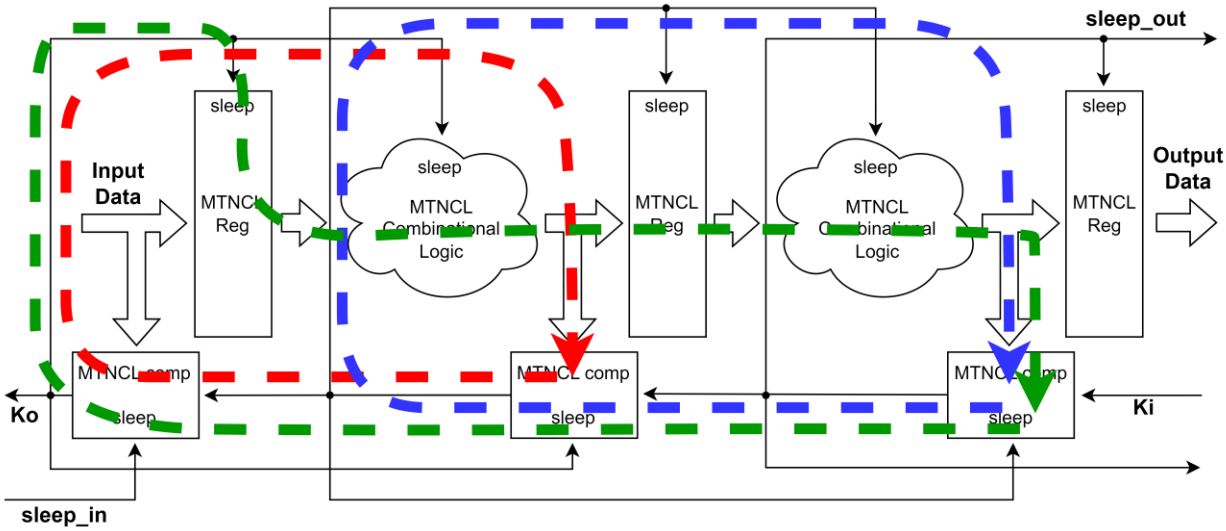

Figure 22: MTNCL Architecture Combinational Feedback Loops

```
// Break combinational loops in completion detection units
set_disable_timing -from a -to z [get_cells -filter {ref_lib_cell_name =~ th22n?*}]
```

Figure 23: SDC Command for Breaking Timing Loops

The *only* exception is the input completion detection's *TH22n* as it does not form any loops, and its *a*-to-*z* timing arc aids in constraining the input interface. Although MTNCL register cells are traditionally constructed using *TH12* cells with external feedback, this work utilizes a custom cell with internal combinational feedback. The intrinsic feedback path is easier to break, unburdens the placement and routing tool from the necessary routing, and yields more consistent latching behavior. The Liberty transformations detailed in Section 3.1 were applied, eliminating the need to explicitly break the feedback paths with SDC commands during physical implementation.

3.3.1 MTNCL Timing Constraints for Reliability

The remainder of this section closely tracks Section 2.5 by providing pairs of constraints that directly address and resolve each of the four relative timing assumptions introduced. In response to the relative nature of the timing assumptions, each pair consists of one max delay constraint for the event that must occur first and one min delay constraint that guarantees the second event occurs last. Where possible, the start point of each constraint is set to the output of a completion component *TH22n* to facilitate comprehension. Pseudocode is provided for each constraint, closely resembling the industry-standard SDC format to assist with easier adoption by circuit designers. Note that, due to the terminal inverter after the *TH22n*, some of the presented SDC options might seem inverted, e.g., a request for DATA would correspond to the fall of the *TH22n* output.

The first pair of constraints targets the DATA completion race condition by ensuring that the complete DATA wave always arrives at the input of the register before the adjacent completion component acknowledges it. Figure 24 and Figure 25 provide a visual representation of the two paths and the associated SDC pseudocode, respectively. *Path A* begins with a request for DATA from the *TH22n* in *Completion Component (i)*, propagates backward to the *Ki* of the previous completion component, unsleeps *Reg(i-1)* and *Combinational Logic(i-1)*, and ultimately results in a DATA wave arriving at the input of *Register (i)*.

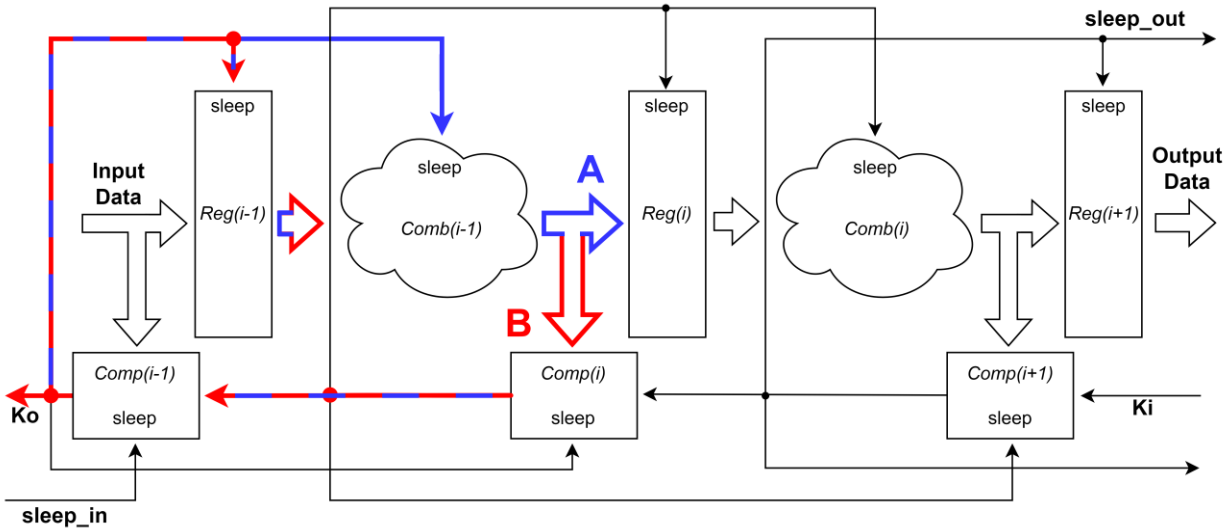

Figure 24: MTNCL DATA Completion Timing Constraint Paths

```
// A < B
set_max_delay A
        -fall_from      [ get_pins “Comp(i)/th22n/z” ]
        -to             [ get_pins “Reg(i)*/a” ]
        -fall_through   [ get_pins “Reg(i-1)*/sleep Comb(i-1)*/sleep” ]
        -rise

set_min_delay B
        -fall_from      [ get_pins “Comp(i)/th22n/z” ]
        -to             [ get_pins “Comp(i)/th22n/a”]
        -fall_through   [ get_pins “Reg(i-1)*/sleep” ]
        -rise
```

Figure 25: MTNCL DATA Completion Timing Constraints

Note that while DATA wave propagation always includes the rise propagation arc through the sleep pin of registers, the bottleneck to DATA wave propagation might alternatively occur in a combinational logic cell with DATA at its inputs but waiting for *sleep* to deassert. Therefore, the constraint’s *through* parameter includes the combinational logic sleep pins.

The two paths mostly overlap but have a critical difference. Unlike *Path A*, *Path B* does not flow through the sleep pins of *Combinational Logic (i-1)*. If these pins were included, the *set_min_delay* constraint would capture very short, false paths through the sleep pins of the last

level of logic, before DATA has fully propagated through the combinational logic. These non-existent paths would obstruct timing closure. *Path B* diverges from *Path A* at the output of the combinational logic by flowing through the completion detection and terminating at the input of the original *TH22n*.

The next pair of constraints guarantees that DATA waves latch in the register before subsequent NULL waves arrive from the upstream stage, thereby solving the DATA handshaking race condition. Figure 26 showcases a visual representation of the paths, and the SDC pseudocode is provided in Figure 27. *Path C* begins with a request for NULL from the *TH22n* in *Completion Component (i)*, unsleeps *Register (i)*, and ends with the register output rising. *Path D*, on the other hand, flows through the *Ki* input of the upstream completion, asserts the *sleep* pins of the upstream register and combinational logic, and ends with the fall of *Register (i)'s* inputs. This path accounts for the shortest path to the register input, whether it be NULL propagating from the upstream register through the combinational logic or *sleep* driving the last stages of the combinational logic.

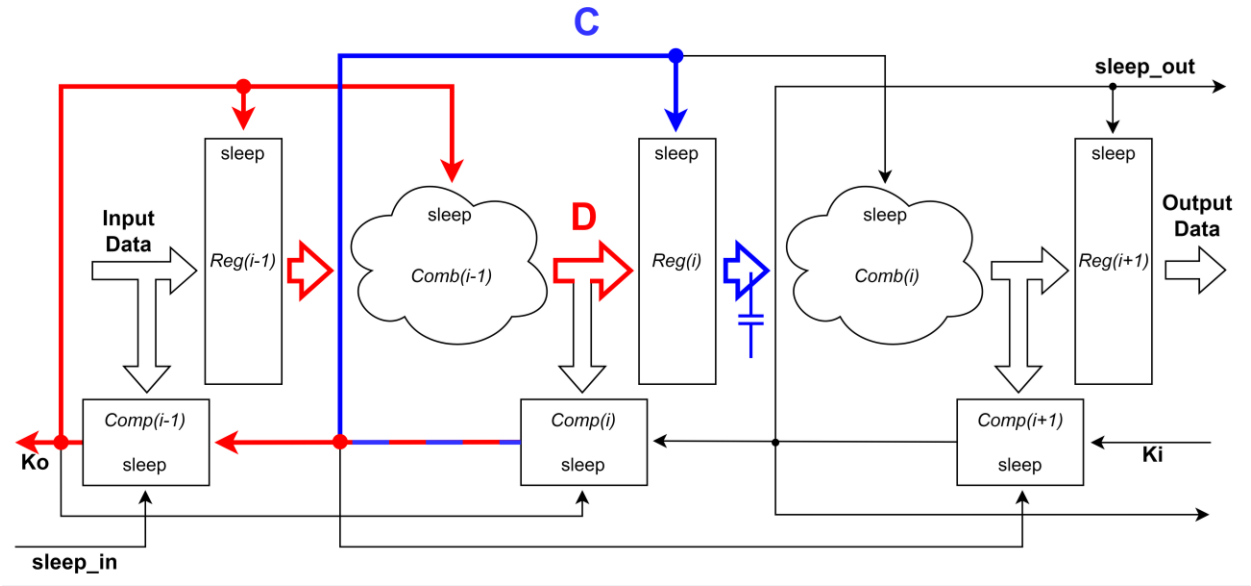

Figure 26: MTNCL DATA Handshaking Timing Constraint Paths

```
// C < D
set_max_delay C
        -rise_from      [ get_pins “Comp(i)/th22n/z” ]
        -to             [ get_pins “Reg(i)*/z” ]
        -fall_through   [ get_pins “Reg(i)*/sleep” ]
        -rise

set_min_delay D
        -rise_from      [ get_pins “Comp(i)/th22n/z” ]
        -to             [ get_pins “Reg(i)*/a”]
        -rise_through   [ get_pins “Reg(i-1)*/sleep Comb(i-1)*/sleep” ]
        -fall
```

Figure 27: MTNCL DATA Handshaking Timing Constraints

The third relative timing assumption is handled by the constraints, depicted in Figure 28 and Figure 29, which ensure that the entire stage transitions to NULL before the completion detection acknowledges the NULL wave. Both paths begin with a request for NULL to the previous completion component but diverge when the upstream completion asserts its *Ko* signal. *Path E* constrains the maximum delay for every gate in the stage to transition to NULL which includes the register, combinational logic, completion detection, and current register input.

Meanwhile, *Path F* ensures that the output of the completion detection does not acknowledge NULL until the entire stage has transitioned.

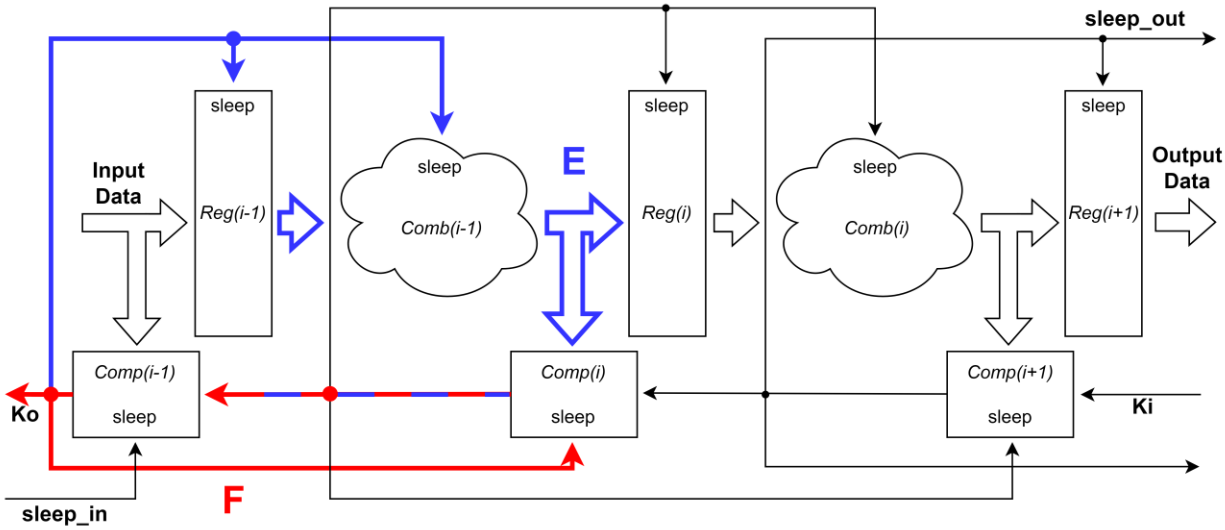

Figure 28: MTNCL NULL Completion Timing Constraint Paths

```
// E < F
set_max_delay E
        -rise_from      [ get_pins “Comp(i)/th22n/z” ]
        -to             [ get_pins “Reg(i-1)*/z Comb(i-1)*/z Comp(i)*/z Reg(i)*/a” ]
        -rise_through   [ get_pins “Reg(i-1)*/sleep Comb(i-1)*/sleep Comp(i)*/sleep” ]
        -fall

set_min_delay F
        -rise_from      [ get_pins “Comp(i)/th22n/z” ]
        -to             [ get_pins “Comp(i)/th22n/a”]
        -rise_through   [ get_pins “*/sleep” ]
        -fall
```

Figure 29: MTNCL NULL Completion Timing Constraints

The final pair of constraints, *G* and *H*, solves the NULL handshaking race condition and mirrors *C* and *D*, but with flipped arcs. Figure 30 illustrates the paths, and Figure 31 provides the SDC pseudocode for the pair. *Path G* begins with a request for DATA from *Completion Component (i)*, asserts the *sleep* of the adjacent register, and concludes when register output falls. *Path H* flows upstream, unsleeps the upstream register and combinational logic, and ends

with the arrival of the next DATA wave at the input of *Register (i)*. Like *Path B*, the *sleep* pins of the combinational logic were deliberately excluded, guaranteeing that the oncoming DATA wave flows through the combinational logic.

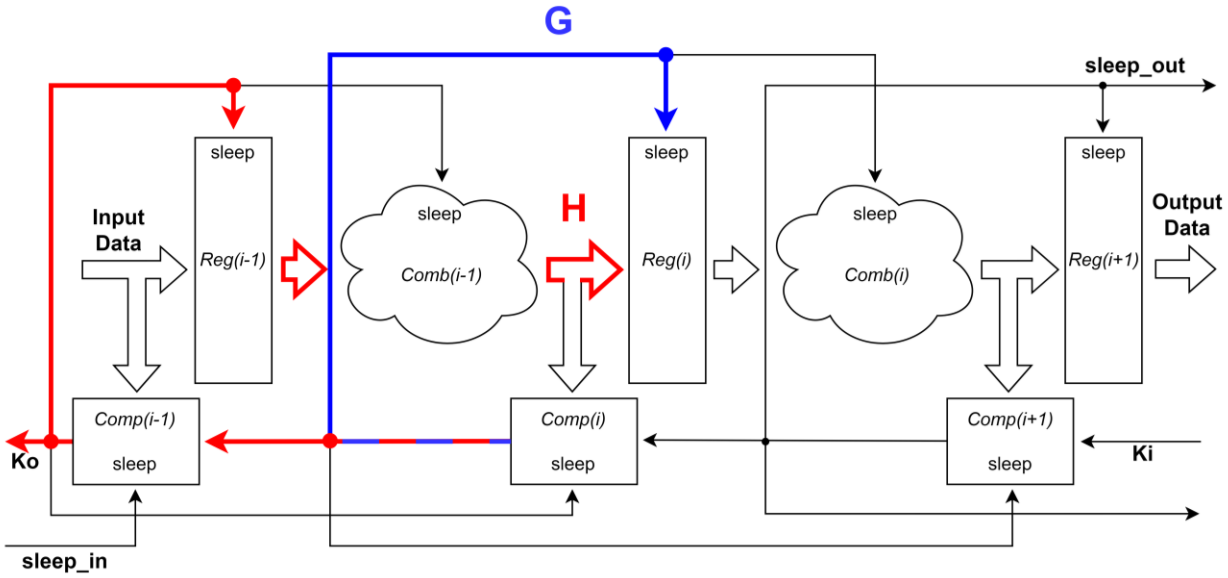

Figure 30: MTNCL NULL Handshaking Timing Constraint Paths

```
// G < H
set_max_delay G
        -fall_from      [ get_pins “Comp(i)/th22n/z” ]
        -to             [ get_pins “Reg(i)*/z” ]
        -rise_through   [ get_pins “Reg(i)*/sleep” ]
        -fall

set_min_delay H
        -fall_from      [ get_pins “Comp(i)/th22n/z” ]
        -to             [ get_pins “Reg(i)*/a”]
        -fall_through   [ get_pins “Reg(i-1)*/sleep” ]
        -rise
```

Figure 31: MTNCL NULL Handshaking Timing Constraints

To eliminate all timing assumptions and restrictions, it is essential to consider the primary ports. The designer has two options for preventing timing hazards between blocks. The first is to perform full hierarchical timing analysis using the four constraint pairs across the boundary,

including the output interface of the first block, any logic and routing between the blocks, and the input interface of the subsequent block. The second and simpler option frees the user from complete boundary timing analysis but imposes two requirements. Any logic and routing between the output register and input register must be input-complete and observable, and the straightforward way to satisfy this requirement is to include *no combinational logic* between these registers. The designer has two options for the second restriction—ensure that NULL waves always arrive before the sleep input to the second circuit by constraining the three components of the path or implement the input completion of the second block with NCL threshold gates to make it input-complete with respect to NULL. Having satisfied these requirements, the circuit designer can close timing on both blocks individually.

The output interface—which includes data outputs, *Ki*, and *sleep_out*—has no potential race conditions. The downstream component will issue requests for DATA and NULL independently and receive them asynchronously. The input interface, however, is still vulnerable to all four of the relative race conditions. Fortunately, the constraint pairs provided earlier in this section can be easily adapted to protect against these error scenarios, primarily by modifying the endpoint of the *set_min_delay* constraint to the *Ko* port. If the constraints presented thus far are applied and closed on an MTNCL circuit, that circuit can be trusted to operate reliably in the process, voltage, and temperature (PVT) corners selected by the user during timing signoff. The author has identified no additional timing vulnerabilities in the MTNCL architecture.

3.3.2 MTNCL Timing Constraints for Optimization

Building upon the timing constraints that guarantee the reliable operation of MTNCL circuits, this section introduces an additional set that can be used during physical implementation to optimize an MTNCL circuit for high performance, reduced power consumption, low area, or

some tradeoff among these three. Perhaps the most crucial constraint corresponds to the *Ko* cycle time and is constructed with the same subcycle path presented in Section 2.4. Given the perfect correspondence between the two, Figure 9 has been added below for easy access along with the SDC pseudocode in Figure 32. The initial state of the circuit is provided in blue, while the transitions encompassing the path are indicated in red. The path begins with the completion component issuing a request for DATA, prompting the upstream completion to respond with a *Ko* of '0'. This unsleeps the adjacent register, combinational logic, and the completion detection. The next DATA wave will then propagate through the register and combinational logic, concluding with the original completion detection verifying the complete DATA wave. Note that this path is not a full cycle, i.e., the transition through the final *TH22n* is not included.

This path, henceforth referred to as the *I path*, plays a critical role in assessing the complete cycle delay (T_{DD}), which is the sum of the *I path* delay and the average *TH22n* propagation delay. It begins with a request for DATA and concludes upon receiving and verifying the DATA wave. Thus, assuming no other bottlenecks and a balanced pipelined circuit, Equation 1 can be rewritten as a function of the *I path* delay and average *TH22n* delay (T_{TH22}). Following timing closure, Equation 2 should theoretically provide the designer with a lower bound for the performance of the target circuit. Naturally, MTNCL's completion detection and perfect adaptation to delay variation mean that the final circuit will outperform what Equation 2 estimates. An analysis of the effectiveness of this performance prediction is carried out in Section 5.4.

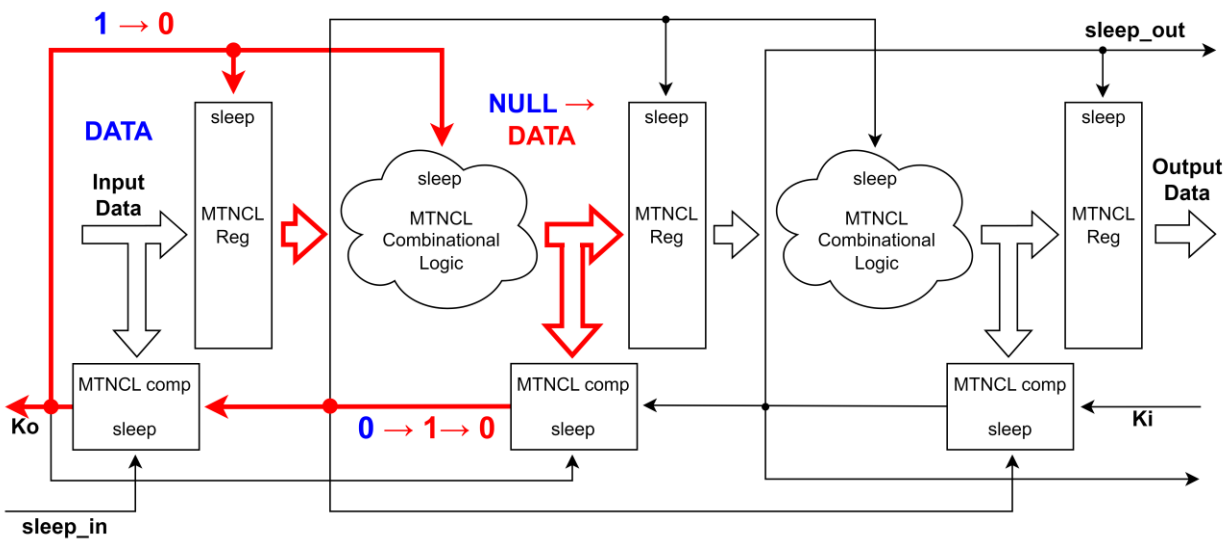

Figure 9: MTNCL DATA Cycle Constraint Path

```
// I
set_max_delay I
        -fall_from      [ get_pins "Comp(i)/th22n/z" ]
        -to             [ get_pins "Comp(i)/th22n/a" ]
        -fall_through   [ get_pins "Reg(i-1)*/sleep Comb(i-1)*/sleep" ]
        -rise
```

Figure 32: MTNCL DATA Cycle Constraint

$$T_{cycle} = 2 * (T_I + T_{TH22}) \quad (2)$$

To reiterate, the provided equation is valid only if no other paths in the design bottleneck the circuit's performance. Therefore, additional constraints must be included for the input and output interfaces of the design. While the four remaining paths presented in this section are unlikely to bottleneck most designs, constraints are applied to guarantee this assumption and ensure the entire circuit is constrained. Figure 33 illustrates the associated paths while Figure 34 presents the SDC pseudocode. *Path J* ensures that DATA and NULL waves travel promptly from the design's ports to the input register. Conversely, *Path K* assures that the data inputs and *sleep_in* result in rapid transitions on the *Ko* port. The output interface is constrained by paths *L* and *M*,

confirming that DATA and NULL waves travel efficiently from the output register to the output ports and that the path from *Ki* to the *sleep* output is short.

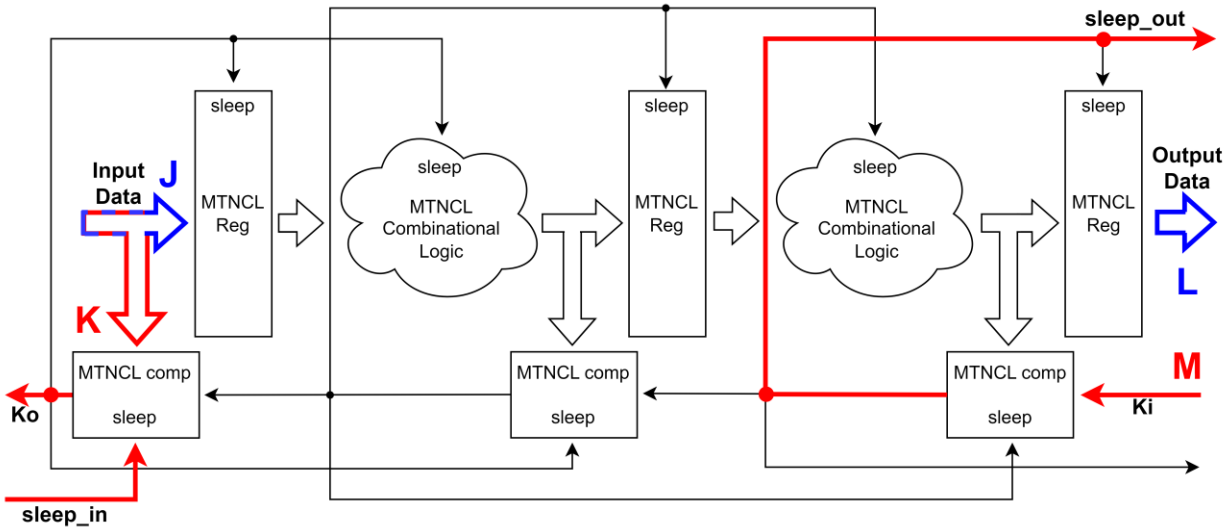

Figure 33: MTNCL Performance-related I/O Constraint Paths

```
set_max_delay J
        -from   [ get_ports -filter {direction == in} ]
        -to     [ get_pins “Reg(0)*/a”]

set_max_delay K
        -from   [ get_ports -filter {direction == in} ]
        -to     [ get_ports ko ]

set_max_delay L
        -from   [ get_pins “Reg(X)*/z” ]
        -to     [ get_ports -filter {direction == out} ]

set_max_delay M
        -from   [ get_ports ki ]
        -to     [ get_ports sleep_out ]
```

Figure 34: MTNCL Performance-related I/O Constraints

Having addressed all the paths in the MTNCL circuit, the constraint values must be selected properly. For reliability, the user must simply set the *min_delay* value greater than or equal to the *max_delay* value, e.g., $A \leq B$. In contrast, the performance delay values should be lower than the

value of the *I path* constraint, which is straightforward to achieve for two reasons. First, the *I path* is longest of the constrained paths, as it passes through the highest number of gates. Second, NULL wave generation via *sleep* is inherently faster than DATA wave propagation, speeding up approximately half of the path types. However, it should be noted that *I* corresponds to the *worst-case* data-dependent delay. The average delay across all operating patterns will be lower, so it is advisable to set the performance path delays at an appropriate margin lower than the *I path* delay. The selected margin should factor in the target circuit and the disparity between its *worst-case* and *average-case* data-dependent delays.

When the circuit under design is intended to be integrated into a larger system, the input and output constraints warrant additional consideration. In this scenario, it is recommended to tightly constrain these paths to mitigate potential issues during timing closure of the block into which the circuit under design is being integrated.

4. EVALUATION SETUP

This section begins with an outline of the cell libraries and associated collateral used for implementation of the test circuits. Following the cell libraries, the physical implementation flow will be covered briefly as it is not a novel contribution of this work. The evaluation circuits and their design approach are then discussed. Lastly, the data collection methods will be outlined.

4.1 Library Preparation

The target technology for the evaluation of this flow is the TSMC 65nm bulk planar general-purpose (GP) technology. While ARM Boolean cells and the provided Liberty and LEF files were used for single-rail synthesis, the final circuits were constructed using a pre-existing library of MTNCL cells designed with skewing as described in [11] for high performance and low power. NCL *THnn* gates were utilized in the completion detection and for implementing the handshaking where necessary. To permit fine-grained optimization, the flow was also provided with a large set of buffers and inverters from the ARM libraries. The MTNCL cells were designed with LVT and HVT transistors, while the NCL cells, buffers, and inverters exclusively utilize LVT transistors.

Following *R+C+CC* parasitic extraction in the *RCmax* corner using Siemens Calibre xRC, all cells were characterized using Synopsys PrimeLib. The corner selected for characterization included the *ss* device corner, a *0.9V* supply voltage, and a *125C* temperature. The *0.9V* supply was selected to account for a 10% IR drop in the power grid following physical implementation. Full state partitioning was enabled to exhaustively characterize the cells across all input combinations, and ECSM Liberty modeling was selected instead of NLDM for greater accuracy. All cells were characterized for timing, power, and noise attributes.

The Abstract Generator utility in Cadence Virtuoso was employed to abstract the cells and generate LEF files. As described in Section 3.1, two sets of cell collateral were produced: an *image* library with the *sleep* pins removed for dual-rail synthesis and the unmodified cells for physical implementation. The fall transition timing, power, and noise arcs were replaced with the rising transition values in the *image* library to enable meaningful optimization during synthesis.

4.2 Physical Implementation

Although the physical implementation flow used in this work is neither novel nor the focus of this dissertation research, it is presented briefly to ensure the entire design flow for the test circuits is documented for reference with the collected data. All physical implementation steps were carried out in Cadence Innovus using a custom set of TCL scripts.

The first step of the flow consisted of reading in all the design collateral, including the Verilog netlist of the design, the Liberty models for the cell libraries, the LEF files with the cell geometry, one or more SDC files with the timing constraints for the associated design, and a Multi-Mode Multi-Corner (MMMC) file that specifies the Liberty, SDC, and PVT corners to be used in timing analysis and optimization.

Following design initialization, the second step entailed preparing the floorplan. A square floorplan with all the I/O pins placed along the top was implemented for each design. The floorplan size was automatically scripted to achieve 70% placement density, calculated based on the cumulative cell area in the input Verilog netlist. When the final density exceeded 85% for some designs, the target density was iteratively reduced to alleviate congestion.

After placing well taps in a checkerboard pattern to meet latch-up DRC requirements, the power grid was constructed, beginning with M1 rails for the *VDD* and *VSS* pins within each cell

macro. Straps were excluded for M2 through M5 to alleviate routing congestion; instead, M6 through M8 were used to construct a mesh for *VDD* and *VSS* with stacked vias from M6 down to the M1 rails. In addition to enhancing the routability of the design, the power grid was designed to minimize IR drop. Beyond the standard goal of ensuring the PVT corners employed in timing analysis accurately constrain the design, as is the case for synchronous circuits, MTNCL can adapt perfectly to dynamic and variable IR drop across the circuit and achieve higher throughput, an additional, unique benefit imparted to MTNCL circuits by high quality power grids.

Prior to placement, the third step of the flow includes configuring a variety of settings to control the remainder of the flow. The optimization power effort is set to *low* for designs targeting *high performance* and *high* for *low-power* oriented circuits, per Cadence's recommendation in the Innovus documentation [35]. Via the *opt_leakage_to_dynamic_ratio* attribute, Innovus is instructed to give equal priority to optimization of dynamic and leakage power. Modern placement and routing utilities support a feature called *Critical Region Resynthesis (CRR)*—where portions of the design are unmapped and resynthesized using the associated synthesis utility, Genus for this flow—to assist with timing closure. Although likely beneficial to synchronous circuit designers, this has the potential to corrupt the functionality of MTNCL designs. For this reason, it is crucial to limit Innovus to cell resizing by setting the *dont_touch* attribute to *size_ok* for all MTNCL cells. However, Innovus should *not* be restricted in its ability to optimize buffers and inverters, including insertion, resizing, and deletion. With this *dont_touch* configuration, inverters are placed in pairs, preserving the logical functionality of the circuit. Interestingly, as of the writing of this dissertation, power consumption in the final circuit is lower if Innovus is forced to downsize all gates in the design prior to placement using

opt_power -force -pre_place instead of beginning placement with the gate sizes selected by Genus. Cell placement and pre-CTS optimization are then carried out using *place_opt_design*.

Naturally, the entire CTS and post-CTS optimization steps are skipped in this flow due to the lack of clock nets, so the flow moves directly to routing. Following routing, two rounds of post-route optimization are performed to refine setup and hold timing, dynamic power, leakage power, and area. A timing report for the design is generated, and the timing constraint values are iteratively adjusted as necessary to close timing and optimize the design for the chosen application. For larger designs, doing preliminary timing closure after placement and before routing results in a lower overall turnaround time. With the introduction of this work's novel timing constraints to the MTNCL physical implementation flow, the tool can extract the routing parasitics and accurately carry out timing analysis. Innovus can recognize the critical paths and optimize the circuit using many of the powerful optimization techniques presented in Section 2.2.

With the placement of the macros and routing of the interconnects finalized, the circuit can be prepared for export. Filler cells are inserted between all the functional cells to solve common DRC errors related to the implant layers, and a TCL procedure is utilized to add labels to all the I/O ports in preparation for LVS. At this stage, the design is complete, so various design files are exported: the final Verilog netlist, GDS of the layout, final timing report, and the Innovus database for quick design restoration. Finally, a custom script is executed to automatically import the Verilog and GDS into Virtuoso and generate the schematic and layout, respectively. Although the flow employed by this work ends here, all standard signoff tasks, including signoff timing verification using the developed timing constraints, can be carried out in preparation for tapeout.

4.3 Timing Closure Approach

Arriving at finalized, precise values for all the timing constraint groups presented in Section 3.3 is a highly iterative process, but most of the designs followed a similar set of steps. Typically, only one or two values were modified in any single optimization iteration to enable a somewhat controlled approach. Most of the constraints can interfere with one another if not set diligently. As an example, to optimize *I* constraints, Innovus will skew the *sleep* tree to prioritize early *sleep* arrival to gates along the critical path, deprioritizing other gates along paths with positive setup slack. The *C* constraint, however, dictates how quickly sleep is distributed to all the gates. Thus, the *C* and *I* constraints will be in contention if they are both set tightly. For this reason, a controlled approach reveals what impact changing a constraint has on the other paths in the design.

Where possible, a single value for a given constraint type, such as the *I* constraint, is used instead of individual values for each stage. This is feasible due to the application of register retiming during the synthesis process and thus the presence of balanced pipeline stages. The initial values for the timing constraints are set to values that are trivial to meet, i.e., very high max delay constraints and very low min delay constraints.

The timing closure process for *high-performance* designs begins with the *I* path delay, as the performance of MTNCL circuits is directly derived from this path (explained in Section 2.4). The *I* delay was set to the lowest value Innovus could achieve without resulting in negative slack. Subsequently, the *A* delay is set to approximately 10% less than *I* to avoid bottlenecking DATA wave propagation. This is easily achieved since the *I* paths are always longer due to the inclusion of the completion detection logic.

Next, the *E* and *G* constraints are set near their lower bounds, though finding the absolute minimum is unnecessary because the corresponding *F* and *H* path delays are often noticeably longer than *E* and *G*, respectively. When preliminary values are selected for *E* and *G*, *F* and *H* can be set to greater than or equal to their twin constraint. If there is contention at this point, the *I* value might need to be loosened slightly to eliminate negative slack. It was observed that setting *E* and *G* close to their lower bound resulted in fewer hold buffers required to fix *F* and *H*, resulting in lower area, active energy, and static power.

At this point, the *C* constraint can be set followed by the *D* constraint, which is defined to be greater than or equal to *C*. Closing the timing again often requires a few iterations, where *C* usually has positive slack while *D* has approximately the same magnitude of negative slack. This issue is resolved by simply reducing both constraints by the magnitude of the slack. It was also observed that setting *C* and *D* too high or too low resulted in a sizable increase in buffering and thus degraded QoR.

The next and last step is to close the reliability and performance constraints corresponding to the input and output interfaces. Keeping all these boundary constraints far lower than *I*, and thus avoiding a bottleneck, is often trivial given the far shorter logic depth. To reiterate, it is recommended to keep the performance-related boundary constraints tight to enable easier timing closure at the next level up in the design hierarchy.

It should be noted that the *B* constraints were not used during physical implementation of the evaluation circuits. Forcing the slowest single DATA pattern to complete and arrive at the register input (*A*) prior to the fastest DATA pattern getting acknowledged by the adjacent completion detection unit (*B*) would likely compromise one of MTNCL's foremost advantages—completion detection with perfect adaptation to delay variability based on input data patterns. In

place of applying the *B* constraint during physical implementation, it is recommended to verify the *A* and *B* timing relationship during signoff STA. As explained in Section 2.5, this relative timing assumption is fairly trivial and the least concerning of the four.

With the introduction of this work's novel set of timing constraints, if timing violations exist after physical implementation, a standard engineering change order (ECO) flow can be employed using a signoff Static Timing Analysis (STA) tool such as Cadence Tempus or Synopsys PrimeTime. When a design team is nearing the tapeout deadline, adding delay to backward flowing *Ko* paths can close any reliability-related timing assumptions by directly adding delay to the *min delay* paths. Although the performance of the design will likely be diminished in exchange, it remains an option for very fast timing closure of the reliability-related paths.

4.4 Data Collection Strategy

Excluding minimum density DRC errors which are only meaningful to a full tapeout, all designs were fully DRC and LVS clean. *R+C* parasitic extraction in the *SigRCmax* corner was then carried out via Siemens Calibre xRC, followed by *transistor-level simulation* using Cadence Virtuoso and Cadence Spectre. The simulation was configured to use the *ss* corner in the device models and a full *1.0V* supply voltage. With the grid resistance extracted, the simulation calculates the highest fidelity dynamic and variable IR drop. These *post-layout transistor-level simulations* enabled acquisition of performance, active energy per operation, and leakage power metrics at the highest possible accuracy.

The average throughput was measured using a sample size of 20 *Ko* cycles. Active energy per operation was then calculated by integrating the current through the design's *VDD* pin across the same 20 operations. Dividing the total charge by 20 yielded the magnitude of both the charge and active energy per operation, given the *1.0V* supply voltage. This sample size produces

sufficient accuracy for dual-rail designs given their relatively constant switching activity factor. The energy-delay product was calculated to enable an accurate assessment of the tradeoff between performance and active energy.

The reset pin was held high for *100 ns* after power up to allow the leakage current in the design to settle to its equilibrium. Keeping reset asserted generates an accurate idle circuit model, when each stage is asleep and requesting for DATA. The *100 ns* delay was chosen after simulating a sweep across several orders of magnitude and observing that the leakage current had sufficiently equilibrated by this point. Recording the magnitude of the current flowing into the *VDD* pin of the design right before reset deasserted yielded the magnitude of the leakage power, given the *1.0V* supply. Additionally, the area of each circuit was recorded using the *report_area* command in Innovus.

With the values of the *I* constraints for each circuit and with the assistance of Equation 2, the performance of each design was projected. Having collected the actual throughput obtained from simulation, the predicted value was compared and analyzed relative to the actual performance. Finally, any evaluation circuit requiring a logical modification or a modified simulation environment for proper functionality will be discussed in Section 5.5.

4.5 Evaluation Circuit Categories

Three distinct categories of circuits were designed to evaluate the advantages and disadvantages of the proposed synthesis and physical implementation flow. The first category, the baseline, was designed structurally, as has been done historically for MTNCL circuits. Following flattening, a custom script buffered the structural circuits based on each net's load capacitance. The load capacitance for each cell's pins was sourced from the Liberty model after characterization, and each net's load was the summation of its fanout pins in the flattened netlist.

Table 5 lists the buffers along with the associated maximum load capacitance values provided to the script. The buffer names follow the ARM naming convention, so *BUF_X1B_A12TL* and *BUF_X2B_A12TL* have an output inverter with *one* and *two* maximum width PMOS fingers, respectively. The max capacitance of the smallest buffer approximately corresponds to a fanout of five medium-sized MTNCL gates. Following buffering, the structural circuits were physically implemented without timing constraints and will be referred to as *structural*.

Table 5: Buffers Utilized for Structural Circuits

Buffer Name	Max Load Cap (fF)	Approx. Max Fanout
BUF_X1B_A12TL	6	5
BUF_X2B_A12TL	12	10
BUF_X4B_A12TL	24	20
BUF_X8B_A12TL	48	40
BUF_X16B_A12TL	96	80

The second set of circuits, referred to as *synthesized*, was synthesized with the proposed flow using synchronous RTL and physically implemented without timing constraints. Buffering for this set was handled during physical implementation by constraining the max transition for all nets. The *low-power* circuits were constrained to a max transition time of *300 ps* based on the ARM libraries of this technology at the same PVT corner, minimizing buffering and thus power consumption. The max transition constraint for the *high-performance* designs was configured to *110 ps*. It was observed that most MTNCL gates in the library for this technology could achieve this transition time in timing reports when upsized to their maximum, and the value was also set high enough to avoid an explosion in the number of drivers in the adder circuits which could compromise PPA. The faster transition times produced by Innovus in response to this constraint naturally reduced the delays in the target circuit. Conversely, in the absence of timing

constraints, Innovus defaults to downsizing all gates, increasing delays. However, with the goal of maintaining the MTNCL cell sizing selected by Genus and associated optimization to the critical paths, Innovus was limited to buffering by setting the *dont_touch* attribute for all MTNCL cells to *true*.

The final set, referred to as *optimized*, was synthesized using the proposed flow and physically implemented with the timing constraints presented in this work. Naturally, the buffering of this set was done during placement and routing to meet the timing constraints and satisfy the max transition attribute in the Liberty models. A comparison between the first two sets demonstrates the improvement achieved through a robust synthesis flow, while a comparison between the last two sets highlights the advantages of using timing constraints during physical implementation.

Three circuit types are used in the evaluation: 1-stage 64-bit adders, 32×32 Montgomery modular multipliers with varying degrees of pipelining, and AES-256 cores. Adder circuits are ubiquitous in digital circuits and demonstrate that synthesis using a commercial utility and physical implementation with timing constraints can yield significant PPA improvements, even for simple circuits like adders. The Montgomery modular multiplier, which is heavily utilized in cutting-edge privacy preserving technologies like Fully Homomorphic Encryption (FHE) and Zero-Knowledge Proofs (ZKP) [36, 37], provides a comparison for a sufficiently large combinational circuit and demonstrates the advantages of register retiming. Finally, AES continues to be the standard for secret key encryption [38], and the AES-256 core yields an analysis of a logically complex and sequential design. Each circuit category and circuit type include a *low-power* implementation and a *high-performance* implementation. High performance

refers to tight timing constraints, while low power entails loose timing constraints, allowing Genus and Innovus to absorb the positive slack through power and area optimizations.