5. RESULTS AND ANALYSIS

The first three sections of this chapter present the results and analysis for the implemented adders, Montgomery modular multipliers, and AES-256 cores, respectively. Each section is subdivided into three subsections: a high-level description of the circuit architecture, a focus on the *low-power* designs, and an exploration of the *high-performance* designs. The latter two subsections begin with a description of the implementation details of the constituent designs and end with the collected data and associated analysis. The fourth section of the chapter is devoted to the feasibility of estimating circuit performance using the constraint values finalized during timing closure. Finally, this chapter concludes with a discussion of reliability observations made during the implementation of the evaluation circuits.

Unless otherwise specified, all paths included in this chapter are the *worst-case*, maximum delay path. Where necessary, the *Timing Points* have been truncated to maintain legibility. In addition, any negligible discrepancies between the total area presented in the summary tables and the cell composition tables are due to the exclusion of a handful of NCL *TH22* completion detection cells and occasional *TIE* cells necessary for constants *1* and *0*. Almost all timing paths and cell composition tables referenced in this chapter are far too extensive to add within the document, thus supplementary files have been attached with this document for reference.

5.1 Adders

5.1.1 Architecture

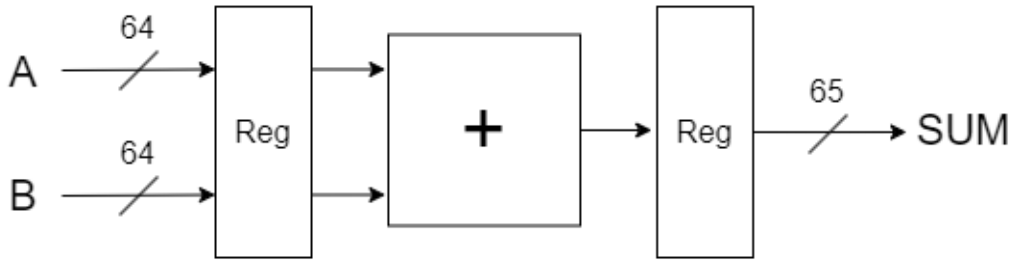

Figure 35: 64-bit Adder Circuit Architecture

Figure 35 illustrates the high-level structure of the 64-bit adder. The two 64-bit inputs are captured by the input register before entering the combinational logic, which varies in architecture across the evaluated designs. The combinational logic produces a 65-bit sum that passes through the output register before exiting the design.

5.1.2 Low Power

5.1.2.1 Implementation

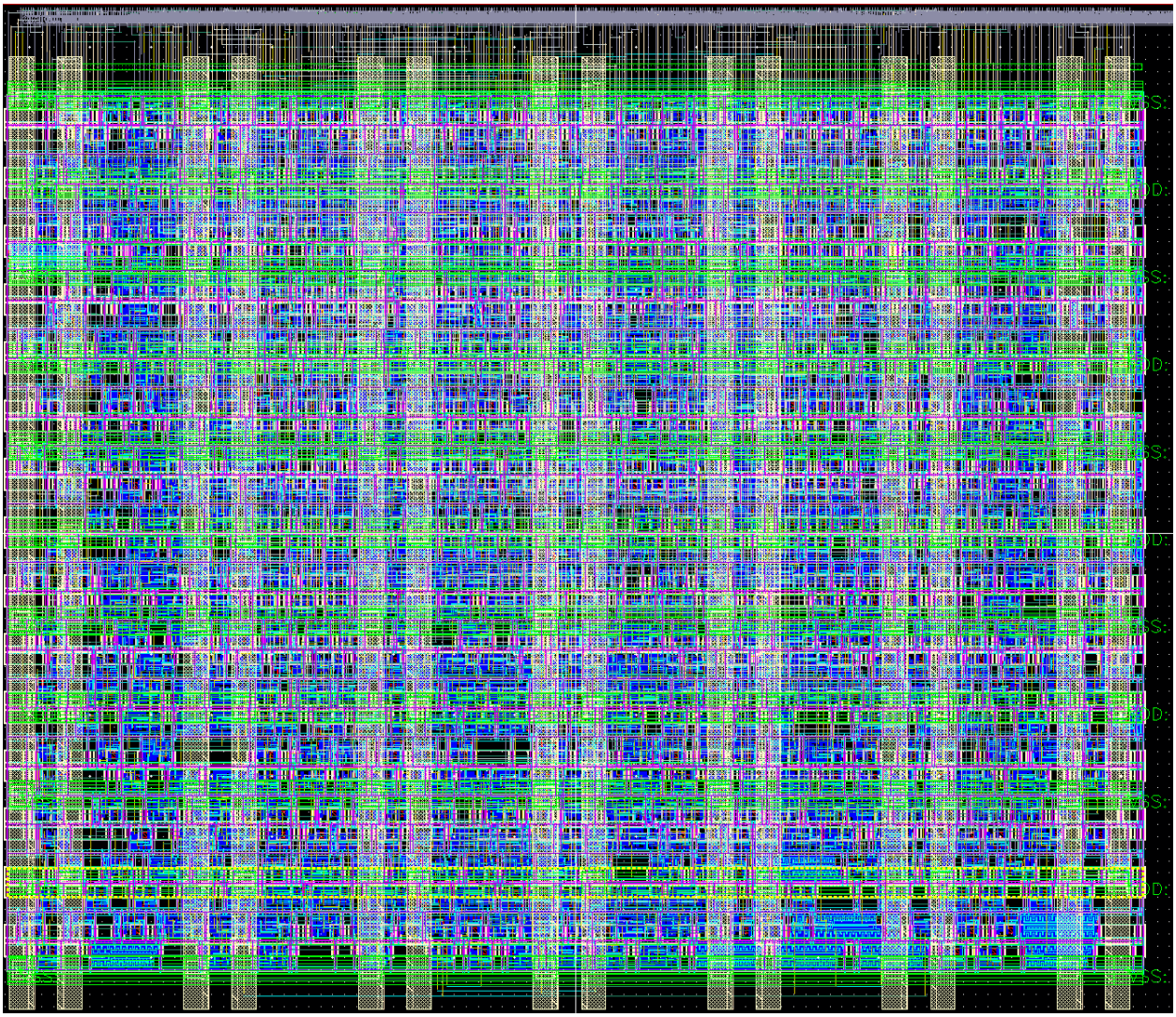

Figure 36: Structural RCA Layout

The *structural low-power* 64-bit adder employs an RCA for the combinational logic, consisting of *TH23* cells for the carry chain and *TH34w2* cells for the sum logic. Although traditionally very slow in synchronous design due to the worst-case scenarios, the RCA is known for its energy and area efficiency. Furthermore, tight project deadlines might preclude the selection and structural creation of more complex adder architectures, making the RCA a

practical selection. The structural completion detection logic is a balanced tree, resembling Figure 20, with a depth of four gates in the *AND tree*. To reduce power consumption, all MTNCL gates in the design were uniformly minimized, a necessary approach when designing structurally and in the absence of timing constraints.

In the RTL utilized with the proposed flow to generate the netlist of the *synthesized* implementation, inputs were summed behaviorally, allowing Genus the flexibility to select its preferred adder architecture. A very slow *16 ns* clock, resulting in several nanoseconds of positive slack, was deliberately chosen for single-rail synthesis to enable Genus to optimize for *low-power*. Figure 37 provides the critical path in the timing report generated during synthesis. Though truncated for brevity, it shows a long carry chain constructed using *TH23* cells, indicating Genus also opted for an RCA. Following the *TH34w2*, which generates the sum bit at the end of the carry chain, is a fairly balanced completion detection *AND tree* constructed with *TH44* gates. The path also indicates that Genus downsized all the MTNCL gates to reduce power consumption.

```
#-------------------------------------------------------------------------------------
#                        Timing Point          Arc   Edge             Cell
#-------------------------------------------------------------------------------------
  reg1_out_reg[129]/CK                         -     R     (arrival)
  reg1_out_reg[129]/Q                          CK->Q F     DFFQ_X0P5M_A12TL
  block0_add_27_34_g1594617_rail1/z            b->z  F     th22m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1592783_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1591526_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1590428_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1589319_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1588260_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1587107_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1586398_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1585477_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  …
  block0_add_27_34_g1537680_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1536783_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1535526_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1534428_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1533319_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1532260_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1531107_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1530398_cout_temp_rail1/z  a->z  F     th23m_LVT_HVT_X0P5N_skewed
  block0_add_27_34_g1530398_sum_rail0/z        a->z  F     th34w2m_LVT_HVT_X0P5N_skewed
  g4252/z                                      c->z  F     th54w22m_LVT_HVT_X0P5N_skewed
  g4251/z                                      a->z  F     th44m_LVT_HVT_X0P5N_skewed
  g4250/z                                      a->z  F     th44m_LVT_HVT_X0P5N_skewed
  g4249/z                                      a->z  F     th44m_LVT_HVT_X0P5N_skewed
  reg2_out_reg[130]/D                          -     F     DFFQ_X0P5M_A12TL
#-------------------------------------------------------------------------------------
```

Figure 37: Synthesized Adder Carry Chain

Since the *synthesized* and *optimized* low-power adders were physically implemented using the same gate-level synthesized netlist, their sole difference was the application of timing constraints during placement and routing. Following the same strategy, the *I Path* constraint was set to a very slow *13.5 ns*, resulting in nearly *4 ns* of positive slack. Innovus responded to the relaxed constraints by keeping all the gates along the *I path* at their minimum sizes.

5.1.2.2 Structural vs. Synthesized

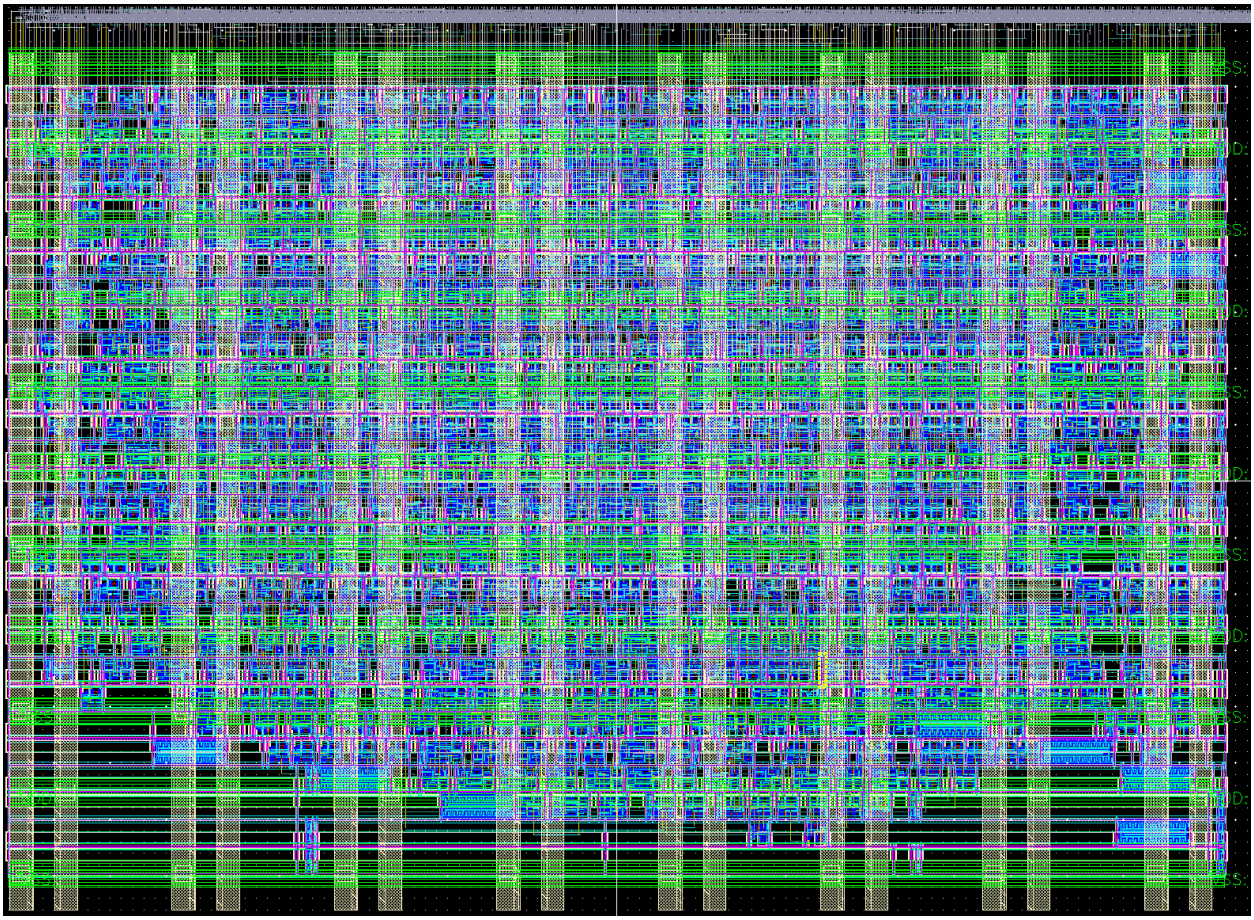

Figure 38: Synthesized Low-power Adder Layout

Table 6: Structural vs. Synthesized Low-power Adder Metrics

	Structural	Synthesized	Percent Improvement
Average Frequency (MHz)	432.9	303.2	-30.0%
Average Active Energy per Operation (pJ)	5.07	4.62	8.9%
EDP (pJ*ns)	11.71	15.22	-30.0%
Leakage Power (µW)	3.328	2.937	11.7%
Area (µm²)	4603	4516	1.9%

Table 6 compares the PPA of the *structural* and *synthesized low-power* adders. Despite the same adder architecture, the *synthesized low-power* adder has a 30% throughput reduction when compared with the *structural* alternative. Although initially unintuitive, the difference in the performance is revealed by examining the delay along *I paths* in each design, presented in Figure 39 and Figure 40. The path and gate delays through the combinational logic and completion detection are very similar, so the key difference is the buffering of the *sleep* signal. The *structural* design, buffered by the custom script, utilizes two buffers: one moderately sized and one very large. The combination of the larger buffers and low fanout of the *TH22* cell results in a quick *sleep* distribution and an overall faster path at *2,172 ps*. The *synthesized* adder, on the other hand, was buffered by Innovus according to a max transition constraint of approximately *300 ps* and employs two very small inverters for buffering, resulting in the *TH22* cell bearing significant fanout. As a result, the time required for *sleep* distribution is substantially longer, and the overall *I path* delay is significantly larger than the equivalent path in the *structural* RCA.

```
#-------------------------------------------------------------------------------------------
# Timing Point  Arc   Edge  Cell                        Fanout   Trans     Delay    Arrival
#                                                                (ps)      (ps)       (ps)
#-------------------------------------------------------------------------------------------
  ki            ki    F     (arrival)                        1  50.000    34.000     34.000
  out_regth22n  b->z  R     th22n_with_inv_LVT_X0P7N         2  50.000   223.700    257.700
  sleep_oLevel  A->Y  R     BUF_X4B_A12TL                   18  43.000    68.200    325.900
  sleep_oLevel  A->Y  R     BUF_X16B_A12TL                 114  24.400    71.500    397.400
  in_reg_h22n/  b->z  F     th22n_with_inv_LVT_X0P7N         3  33.600   202.700    600.100
  ko_Buff1/Y    A->Y  F     BUF_X4B_A12TL                    2  38.100    57.900    658.000
  ko_Buff4/Y    A->Y  F     BUF_X16B_A12TL                 114  13.500    60.300    718.300
  in_reg_/z     s->z  R     th12m_LVT_HVT_X0P5N_skewed       3  27.200    75.700    794.000
  rca_add_cout  b->z  R     th23m_LVT_HVT_X0P5N_skewed       3  55.200   121.500    915.500
  rca_add_cout  a->z  R     th23m_LVT_HVT_X0P5N_skewed       3  50.700   107.700   1023.200
  rca_add_cout  a->z  R     th23m_LVT_HVT_X0P5N_skewed       3  44.200   132.100   1155.300
  rca_add0_cou  a->z  R     th23m_LVT_HVT_X0P5N_skewed       3  57.600   118.600   1273.900
  rca_add1_cou  a->z  R     th23m_LVT_HVT_X0P5N_skewed       3  52.100   117.600   1391.500
  rca_add2_cou  a->z  R     th23m_LVT_HVT_X0P5N_skewed       3  52.900   125.500   1517.000
  rca_add3_sum  b->z  R     th34w2m_LVT_HVT_X0P5N_skewed     2  54.000   153.900   1670.900
  out_regGs1_6  d->z  R     th24compm_LVT_HVT_X0P5N_skewed   1  53.700    96.400   1767.300
  out_regGco_G  c->z  R     th44m_LVT_HVT_X0P5N_skewed       1  25.800   113.800   1881.100
  out_regGco_G  b->z  R     th44m_LVT_HVT_X0P5N_skewed       1  31.000   144.000   2025.100
  out_regGco_G  a->z  R     th33m_LVT_HVT_X0P5N_skewed       1  44.700   147.500   2172.600
  out_regth22n  a     R     th22n_with_inv_LVT_X0P7N         1  62.900     0.000   2172.600
#-------------------------------------------------------------------------------------------
```

Figure 39: Structural Low-power Adder Average DATA Cycle Path

```
#-------------------------------------------------------------------------------------------
# Timing Point  Arc   Edge  Cell                        Fanout    Trans     Delay    Arrival
#                                                                 (ps)      (ps)       (ps)
#-------------------------------------------------------------------------------------------
  ki            ki    F     (arrival)                        1   50.000     0.200      0.200
  block0_reg_nz b->z  R     th22n_with_inv_LVT_X0P5N        37   50.000   445.400    445.600
  comp1_inst_c  b->z  F     th22n_with_inv_LVT_X0P5N        49  252.500   418.300    863.900
  FE_OFC0_ko/Y  A->Y  R     INV_X0P8B_A12TL                  9  187.400   190.000   1053.900
  FE_OFC2_ko/Y  A->Y  F     INV_X0P8B_A12TL                 71  116.800   332.200   1386.100
  block0_reg_G  s->z  R     th12m_reg_LVT_HVT_X0P5N_skewed   2  244.700   143.000   1529.100
  block0_core_  b->z  R     th23m_LVT_HVT_X0P5N_skewed       3  120.200   144.300   1673.400
  block0_core_  a->z  R     th23m_LVT_HVT_X0P5N_skewed       3   90.600   122.700   1796.100
  block0_core_  a->z  R     th23m_LVT_HVT_X0P5N_skewed       3   86.100   123.600   1919.700
  block0_core_  a->z  R     th23m_LVT_HVT_X0P5N_skewed       3   87.600   117.800   2037.500
  block0_core_  a->z  R     th23m_LVT_HVT_X0P5N_skewed       3   82.700   121.600   2159.100
  block0_core_  a->z  R     th23m_LVT_HVT_X0P5N_skewed       3   86.600   120.200   2279.300
  block0_core_  a->z  R     th34w2m_LVT_HVT_X0P5N_skewed     2   84.800   126.700   2406.000
  block0_core_  b->z  R     th24compm_LVT_HVT_X0P5N_skewed   1   81.900   137.000   2543.000
  block0_core_  b->z  R     th44m_LVT_HVT_X0P5N_skewed       1   83.200   193.100   2736.100
  block0_core_  a->z  R     th44m_LVT_HVT_X0P5N_skewed       1  109.500   145.500   2881.600
  block0_core_  a->z  R     th44m_LVT_HVT_X0P5N_skewed       1  101.600   130.700   3012.300
  block0_reg.t  a     R     th22n_with_inv_LVT_X0P5N         1   60.000     0.000   3012.300
#-------------------------------------------------------------------------------------------
```

Figure 40: Synthesized Low-power Adder Average DATA Cycle Path

Further, the presented paths correlate closely with the measured performance of both designs. The average operating period of the *structural* adder was *2.31 ns*, while the period of the *synthesized* adder was *3.29 ns*—42% longer. Notably, the delay through the path in the *synthesized* design was *3.012 ns*, 39% longer than the delay through the *structural* design. These two paths were carefully selected to include exactly six carry propagations, indicated by the presence of *TH23* cells. On average, the longest carry chain in an *n*-bit RCA is $log_2 n$ for random data patterns; therefore, the path delays presented with a carry chain of six provide accurate models for the average performance of the adder circuits [4, 39].

The average active energy per operation, leakage power, and area can be understood by examining the types and number of cells used in constructing the *structural* and *synthesized* adders, tabulated in Table 7 and Table 8, respectively. Unsurprisingly, the cell type and instance count of the MTNCL gates are almost identical given that both designs implement the core adder with an RCA. The number of *TH12* cells match almost perfectly since these cells are used to implement the registers, while the *TH23* and *TH34w2* cells implement the full adders in the combinational logic. Finally, *TH24comp* and *TH44* compose the completion detection logic, and the remaining inverter and buffer cells implement the *sleep* trees.

Table 7: Structural Low-power Adder Cell Composition

Cell	Instance	Area (μm²)
BUF_X16B_A12TL	7	97.4
BUF_X2B_A12TL	2	5.8
BUF_X4B_A12TL	2	8.6
th12m_LVT_HVT_X0P5N_skewed	388	1862.4
th22m_LVT_HVT_X0P5N_skewed	1	4.3
th22n_with_inv_LVT_X0P7N	2	14.4
th23m_LVT_HVT_X0P5N_skewed	126	786.2
th24compm_LVT_HVT_X0P5N_skewed	98	611.5
th33m_LVT_HVT_X0P5N_skewed	1	5.8
th34w2m_LVT_HVT_X0P5N_skewed	126	1028.2
th44m_LVT_HVT_X0P5N_skewed	31	178.6
Total	784	4603.2

Table 8: Synthesized Low-power Adder Cell Composition

Cell	Instance	Area (μm²)
BUF_X0P7B_A12TL	4	7.7
BUF_X1B_A12TL	1	1.9
INV_X0P6B_A12TL	6	8.6
INV_X0P7B_A12TL	2	2.9
INV_X0P8B_A12TL	2	2.9
th12m_LVT_HVT_X0P5N_skewed	3	14.4
th12m_reg_LVT_HVT_X0P5N_skewed	386	1852.8
th22m_LVT_HVT_X0P5N_skewed	3	13.0
th22n_with_inv_LVT_X0P5N	2	14.4
th23m_LVT_HVT_X0P5N_skewed	126	786.2
th24compm_LVT_HVT_X0P5N_skewed	96	599.0
th34w2m_LVT_HVT_X0P5N_skewed	126	1028.2
th44m_LVT_HVT_X0P5N_skewed	31	178.6
th54w22m_LVT_HVT_X0P5N_skewed	1	5.8
Total	789	4516.3

This difference in *sleep* tree area is more impactful for the active energy of each design. In addition to the sleep tree area being 4.6 times greater in the *structural* design, the cells occupying this area have greater transistor density. For example, whereas *BUF_X16* has *16* full width fingers in the second inverter's PMOS device, the PMOS device in the *INV_X0P6* is only 60% of the maximum width despite occupying the same footprint as a higher density *INV_X1* cell. The *structural* design exclusively incorporates buffers with maximum sized output inverter PMOS devices, while the *synthesized* design heavily utilizes non-maximum sized PMOS devices. Furthermore, the activity factor of the *sleep* signal is always twice that of the data signals, doubling the impact of this discrepancy on active energy. For these reasons, although the area savings through synthesis are only 1.9%, the active energy savings are several times higher at 8.9%. The total cell area dedicated to *sleep* drivers is about *112 µm²* in the *structural* adder and about *24 µm²* in the *synthesized* adder, directly explaining to the area discrepancy between the two.

Of all the design metrics, the *sleep* tree implementation is the most impactful for overall static power dissipation. MTNCL itself targets low leakage power applications through its utilization of gate-level MTCMOS power gating. During idle time, every path between *VDD* and *VSS* includes at least two modest width cutoff transistors, often more depending on the specific threshold gate. Moreover, one or more of these devices are implemented with the higher V_T of the pair. Consequently, MTNCL threshold gates—and MTNCL circuits in general—exhibit very-low static power dissipation.

Conversely, the buffers and inverters leveraged for this work have multiple characteristics that contribute to very high leakage power. First, LVT cells were selected for their higher performance at the expense of static power dissipation. Second, in contrast to MTNCL threshold

gates, the path between *VDD* and *VSS* in inverters consists of a single cutoff device, while buffers naturally contain two such paths in parallel, reducing overall resistance. Finally, depending on the cell, inverters and buffers can have substantially wider transistors. As a result, the total transistor width for each design's sleep tree provides a useful metric for leakage power comparisons. Summing the width indicated by the cell name, *0.6* for *INV*_X0P6 for example, and doubling the width for buffers should approximate the total transistor width of each design's sleep tree. This calculation yields a value of *14.2* for the *synthesized* design and *248* for the *structural* adder, representing a 16-fold increase.

Synthesizing the adder and buffering it with a loose maximum transition constraint, as opposed to designing one structurally and buffering it with a custom script, resulted in approximately 9% less active energy, 11.7% less leakage power, and about 2% less area in exchange for a 30% throughput penalty. Due to this significant throughput drop, the overall EDP also increased by about 30%. However, since the target application for both circuits was low power, the *synthesized* design is ultimately the better choice for applications requiring the least energy per operation and those with significant idle time. The throughput and EDP penalty highlight the tradeoffs involved, which merit careful consideration.

5.1.2.3 Synthesized vs. Optimized

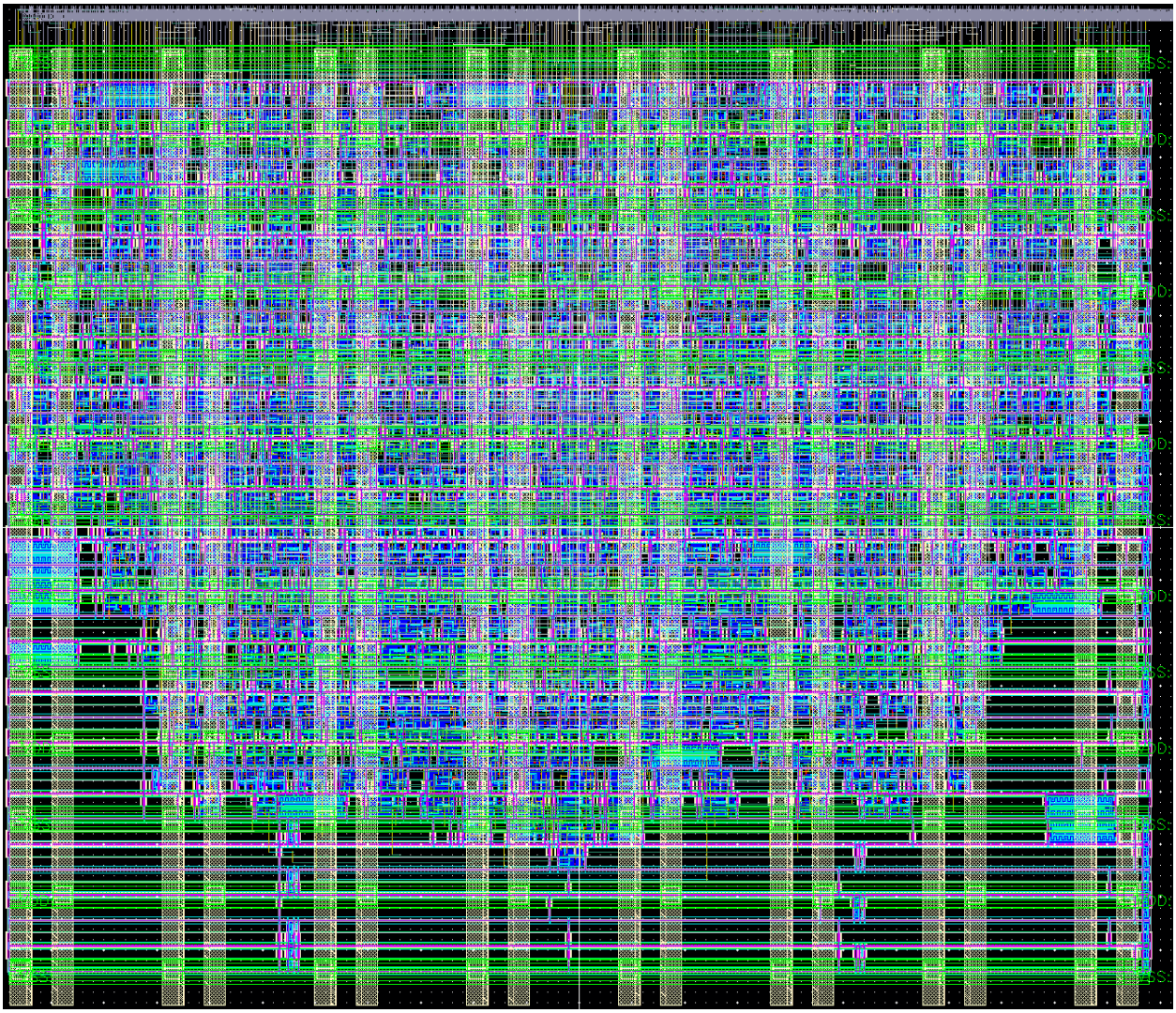

Figure 41: Optimized Low-power Adder Layout

Table 9: Synthesized vs. Optimized Low-power Adder Metrics

	Synthesized	Optimized	Percent Improvement
Average Frequency (MHz)	303.2	351.7	16.0%
Average Active Energy per Operation (pJ)	4.62	4.65	-0.7%
EDP (pJ*ns)	15.22	13.22	13.1%
Leakage Power (µW)	2.937	3.515	-19.7%
Area (µm2)	4516	4541	-0.6%

Table 9 above showcases a comparison of the *synthesized* and *optimized low-power* adders. Figure 40 has been inserted again alongside the same path through the *optimized* adder in Figure 42 for easier comparison. The almost *50 MHz* improvement in the *optimized* design's throughput is a result of the lower delay in the presented path. As was the case for the RCA, the key distinction between the paths is the *sleep* tree implementation. In response to the timing constraints, Innovus more effectively constructed the sleep network, reducing the delay to the register's *sleep* by approximately *400 ps*.

```
#-------------------------------------------------------------------------------------------
# Timing Point  Arc   Edge  Cell                        Fanout    Trans     Delay    Arrival
#                                                                 (ps)      (ps)       (ps)
#-------------------------------------------------------------------------------------------
  ki            ki    F     (arrival)                      1   50.000     0.200      0.200
  block0_reg_nz b->z  R     th22n_with_inv_LVT_X0P5N       37  50.000   445.400    445.600
  comp1_inst_c  b->z  F     th22n_with_inv_LVT_X0P5N       49 252.500   418.300    863.900
  FE_OFC0_ko/Y  A->Y  R     INV_X0P8B_A12TL                 9 187.400   190.000   1053.900
  FE_OFC2_ko/Y  A->Y  F     INV_X0P8B_A12TL                71 116.800   332.200   1386.100
  block0_reg_G  s->z  R     th12m_reg_LVT_HVT_X0P5N_skewed  2 244.700   143.000   1529.100
  block0_core_  b->z  R     th23m_LVT_HVT_X0P5N_skewed      3 120.200   144.300   1673.400
  block0_core_  a->z  R     th23m_LVT_HVT_X0P5N_skewed      3  90.600   122.700   1796.100
  block0_core_  a->z  R     th23m_LVT_HVT_X0P5N_skewed      3  86.100   123.600   1919.700
  block0_core_  a->z  R     th23m_LVT_HVT_X0P5N_skewed      3  87.600   117.800   2037.500
  block0_core_  a->z  R     th23m_LVT_HVT_X0P5N_skewed      3  82.700   121.600   2159.100
  block0_core_  a->z  R     th23m_LVT_HVT_X0P5N_skewed      3  86.600   120.200   2279.300
  block0_core_  a->z  R     th34w2m_LVT_HVT_X0P5N_skewed    2  84.800   126.700   2406.000
  block0_core_  b->z  R     th24compm_LVT_HVT_X0P5N_skewed  1  81.900   137.000   2543.000
  block0_core_  b->z  R     th44m_LVT_HVT_X0P5N_skewed      1  83.200   193.100   2736.100
  block0_core_  a->z  R     th44m_LVT_HVT_X0P5N_skewed      1 109.500   145.500   2881.600
  block0_core_  a->z  R     th44m_LVT_HVT_X0P5N_skewed      1 101.600   130.700   3012.300
  block0_reg.t  a     R     th22n_with_inv_LVT_X0P5N        1  60.000     0.000   3012.300
#-------------------------------------------------------------------------------------------
```

Figure 40: Synthesized Low-power Adder Average DATA Cycle Path

```
#-------------------------------------------------------------------------------------------
# Timing Point       Arc   Edge  Cell                           Fanout      Delay    Arrival
#                                                                           (ps)       (ps)
#-------------------------------------------------------------------------------------------
  ki                 ki    F     (arrival)                         1     12.200     12.200
  block0_comp.th22/z b->z  R     th22n_with_inv_LVT_X1N           72    416.700    428.900
  comp1_inst_comp_   b->z  F     th22n_with_inv_LVT_X0P5N          1    222.100    651.000
  FE_OFC0_ko/Y       A->Y  R     INV_X2B_A12TL                    12     63.000    714.000
  FE_OFC8_ko/Y       A->Y  F     INV_X0P8B_A12TL                  59    267.300    981.300
  block0_reg[8]/z    s->z  R     th12m_reg_LVT_HVT_X0P5N_skewed    2    145.100   1126.400
  block0wrapped_a/z  b->z  R     th23m_LVT_HVT_X0P5N_skewed        3    128.600   1255.000
  block0wrapped_a/z  a->z  R     th23m_LVT_HVT_X0P5N_skewed        3    134.800   1389.800
  block0wrapped_a/z  a->z  R     th23m_LVT_HVT_X0P5N_skewed        3    126.800   1516.600
  block0wrapped_a/z  a->z  R     th23m_LVT_HVT_X0P5N_skewed        3    131.300   1647.900
  block0wrapped_a/z  a->z  R     th23m_LVT_HVT_X0P5N_skewed        3    144.800   1792.700
  block0wrapped_a/z  a->z  R     th23m_LVT_HVT_X0P5N_skewed        3    127.800   1920.500
  block0wrapped_a/z  a->z  R     th34w2m_LVT_HVT_X0P5N_skewed      2    122.200   2042.700
  block0g4384/z      b->z  R     th24compm_LVT_HVT_X0P5N_skewed    1    103.800   2146.500
  block0g4251/z      b->z  R     th44m_LVT_HVT_X0P5N_skewed        1    170.700   2317.199
  block0g4250/z      a->z  R     th44m_LVT_HVT_X0P5N_skewed        1    170.500   2487.699
  block0g4249/z      a->z  R     th44m_LVT_HVT_X0P5N_skewed        1    127.000   2614.699
  block0comp.th22/a  a     R     th22n_with_inv_LVT_X1N            1      0.000   2614.699
#-------------------------------------------------------------------------------------------
```

Figure 42: Optimized Low-power Adder Average DATA Cycle Path

The gate counts for both designs have been included in Table 8 and Table 10 below to facilitate discussion of the remaining metrics. Innovus employed a variety of different cell types in the *optimized* design to apply fine grain improvements, including high-speed buffers (*BUFH*), regular buffers, inverters, and delay cells. Based on the previously described approximation for total transistor width of the *sleep* tree, the *optimized* design featured an area that was 2.5 times the width of the *synthesized* adder, measuring at *36.2*. This substantial increase in area accounts for the *600 nW* rise in leakage power.

Table 8: Synthesized Low-power Adder Cell Composition

Cell	Instance	Area (μm²)
BUF_X0P7B_A12TL	4	7.7
BUF_X1B_A12TL	1	1.9
INV_X0P6B_A12TL	6	8.6
INV_X0P7B_A12TL	2	2.9
INV_X0P8B_A12TL	2	2.9
th12m_LVT_HVT_X0P5N_skewed	3	14.4
th12m_reg_LVT_HVT_X0P5N_skewed	386	1852.8
th22m_LVT_HVT_X0P5N_skewed	3	13.0
th22n_with_inv_LVT_X0P5N	2	14.4
th23m_LVT_HVT_X0P5N_skewed	126	786.2
th24compm_LVT_HVT_X0P5N_skewed	96	599.0
th34w2m_LVT_HVT_X0P5N_skewed	126	1028.2
th44m_LVT_HVT_X0P5N_skewed	31	178.6
th54w22m_LVT_HVT_X0P5N_skewed	1	5.8
Total	789	4516.3

Table 10: Optimized Low-power Adder Cell Composition

Cell	Instance	Area (μm²)
BUFH_X1M_A12TL	1	1.92
BUFH_X3M_A12TL	1	3.84
BUF_X0P7B_A12TL	3	5.76
BUF_X0P7M_A12TL	2	3.84
DLY2_X0P5M_A12TL	2	5.76
DLY4_X0P5M_A12TL	1	5.28
INV_X0P5B_A12TL	3	4.32
INV_X0P6B_A12TL	1	1.44
INV_X0P7B_A12TL	1	1.44
INV_X0P8B_A12TL	5	7.2
INV_X1B_A12TL	3	4.32
INV_X1P4B_A12TL	1	1.92
INV_X2B_A12TL	1	1.92
th12m_LVT_HVT_X0P5N_skewed	3	14.4
th12m_reg_LVT_HVT_X0P5N_skewed	386	1852.8
th22m_LVT_HVT_X0P5N_skewed	3	12.96
th22n_with_inv_LVT_X0P5N	1	7.2
th22n_with_inv_LVT_X1N	1	7.2
th23m_LVT_HVT_X0P5N_skewed	126	786.24
th24compm_LVT_HVT_X0P5N_skewed	96	599.04
th34w2m_LVT_HVT_X0P5N_skewed	126	1028.16
th44m_LVT_HVT_X0P5N_skewed	31	178.56
th54w22m_LVT_HVT_X0P5N_skewed	1	5.76
Total	799	4541.28

The negligible area increase indicates that Innovus effectively utilized the area to close the provided timing constraints. The marginal degradation of the active energy per operation closely correlates with the change in area, while the 13.1% improvement in EDP is a direct result of the *50 MHz* increase in throughput.

The primary objective of the *optimized* adder remains low power; however, it falls short in terms of leakage power. At first glance, the *synthesized* adder appears to be better suited for applications with significant idle time. Nonetheless, the *optimized* adder benefits from timing closure, which ensures reliable operation across the selected PVT corners. Since reliability is paramount and supersedes all four of the measured constraints, the *optimized* design is superior on grounds of reliability alone. Further, with timing closure in additional PVT corners, end users can leverage supply voltage scaling to achieve the same performance as the *synthesized* design while also reducing power consumption.

5.1.2.4 Summary

Table 11: Structural vs. Optimized Low-power Adder Metrics

	Structural	Optimized	Percent Improvement
Average Frequency (MHz)	432.9	351.7	-18.8%
Average Active Energy per Operation (pJ)	5.07	4.65	8.3%
EDP (pJ*ns)	11.71	13.22	-12.9%
Leakage Power (µW)	3.328	3.515	-5.6%
Area (µm²)	4603	4541	1.3%

Table 11 has been included to enable an overall comparison between the baseline *structural* design and the contributions of this research. Innovus was instructed to optimize solely for power through the application of very loose timing constraints. For this reason, the tradeoff of 18.8% less throughput in exchange for an 8.3% reduction in active energy can be considered a success. The leakage power increase of 5.6%, induced by the higher number of buffer cells, is a necessary compromise for reliability, while the EDP degradation of 12.9% is a natural consequence of the

throughput penalty. Additionally, Innovus achieved a 1.3% reduction in the area of the *structural* design.

In general, as will be demonstrated later in this chapter, the only reason why the *structural* design achieves competitive metrics against the *optimized* design is its simplicity. An RCA is an extremely simple design, allowing a designer's manual approach to closely approximate the most optimal implementation provided by modern synthesis and physical implementation tools. Notably, the RCA is the only circuit in this chapter where this unexpected observation holds true.

5.1.3 High Performance

5.1.3.1 Implementation

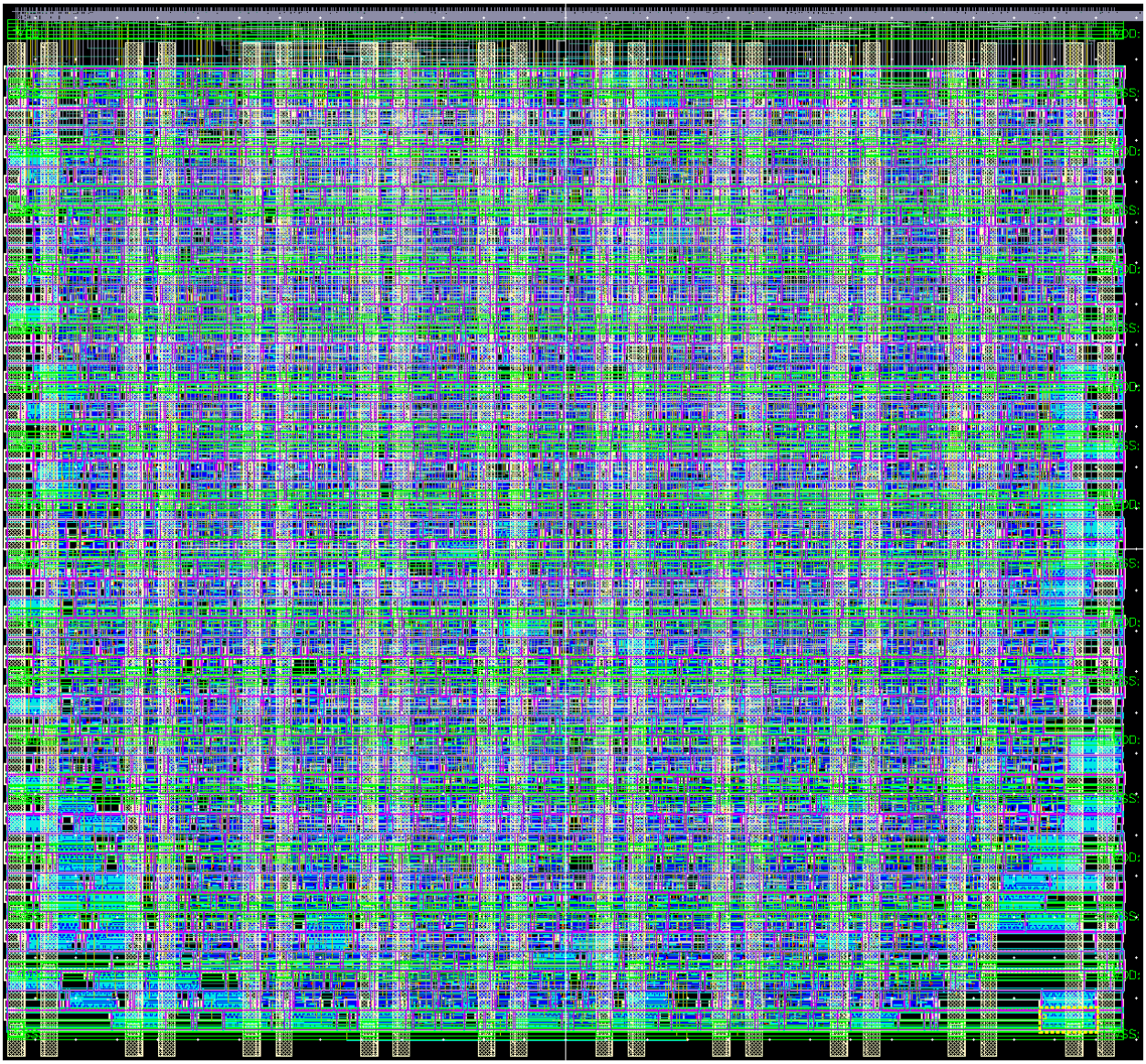

Figure 43: Structural CLA Layout

The combinational logic of the *structural high-performance* 64-bit adder consisted of a standard carry-lookahead adder (CLA). For the structural and low-level details of this adder architecture, please see [4]. Unrolling the carry recurrence of a 64-bit adder results in a substantial reduction in the logic depth when compared with an RCA. For this reason, CLA are

much faster than RCA for all but the smallest widths in a synchronous paradigm, where the *worst-case* full carry chain is the critical path. However, the speedup achieved when moving from an MTNCL RCA to an MTNCL CLA is considerably lower due to MTNCL's completion detection. The *structural* design employs balanced completion detection trees like those used in the RCA. To maximize performance, all the MTNCL gates in the CLA were maximized indiscriminately, which is the only viable option when designing circuits without timing constraints and the ability to identify the critical paths.

All the *low-power* and *high-performance synthesized* adders utilize the exact same behavioral RTL specification. The clock period provided to Genus was *400 ps*, resulting in approximately *50 ps* of negative slack. In an effort to close timing, Genus selected the fastest adder architecture available. Unfortunately, Genus does not specify which adder architecture it selects and attempting to reverse engineer the gate-level netlist is not feasible, though the critical path following dual-rail synthesis indicates that Genus unrolled the carry recurrence. While the logic depth of the *synthesized low-power* adder was *69*, the path in Figure 44 indicates a logic depth of *13* for the *high-performance synthesized* adder. Moreover, after removing the buffer cell and the last three *TH44* gates—which clearly belong to the *AND tree* of the completion detection—the *synthesized* adder logic depth appears to be just *9* gates.

```
#-------------------------------------------------------------------------------
#                     Timing Point          Arc   Edge             Cell
#
#-------------------------------------------------------------------------------
  reg1_out_reg[6]/CK                        -     R     (arrival)
  reg1_out_reg[6]/Q                         CK->Q R     DFFQ_X2M_A12TL
  FE_RC_519_0/z                             a->z  R     th22m_LVT_HVT_X0P7N_skewed
  FE_RC_69_0/z                              c->z  R     th13m_LVT_HVT_X0P7N_skewed
  FE_RC_25_1/z                              a->z  R     th24w22m_LVT_HVT_X0P7N_skewed
  FE_RC_489_0/z                             c->z  R     th54w22m_LVT_HVT_X2N_skewed
  FE_RC_487_0/z                             b->z  R     th34w3m_LVT_HVT_X1N_skewed
  FE_RC_482_0/z                             c->z  R     th54w22m_LVT_HVT_X2N_skewed
  FE_OCPC4_block0_wrapped_adder_ins556_0/Y  A->Y  R     BUF_X16B_A12TL
  FE_RC_136_0/z                             d->z  R     th54w22m_LVT_HVT_X2N_skewed
  FE_RC_218_1/z                             c->z  R     th13m_LVT_HVT_X0P7N_skewed
  FE_RC_500_0/z                             b->z  R     th54w22m_LVT_HVT_X2N_skewed
  FE_RC_544_0/z                             c->z  R     th44m_LVT_HVT_X2N_skewed
  g3869/z                                   d->z  R     th44m_LVT_HVT_X2N_skewed
  g3868/z                                   d->z  R     th44m_LVT_HVT_X2N_skewed
  reg2_out_reg[130]/D                       -     R     DFFQ_X1M_A12TL
#-------------------------------------------------------------------------------
```

Figure 44: Synthesized High-performance Adder Logic Depth

Both the *synthesized* and *optimized* adders were physically implemented using the netlist produced by Genus. The *synthesized* design was buffered by Innovus according to the aforementioned *110 ps* max transition constraint. In contrast, the *optimized* design had the *I path* constrained to *1.275 ns*, resulting in *-15 ps* slack. Innovus responded to this very tight constraint by maximizing the drive strength of most MTNCL cells along the critical path and employing large inverters to construct the *sleep* tree.

5.1.3.2 Structural vs. Synthesized

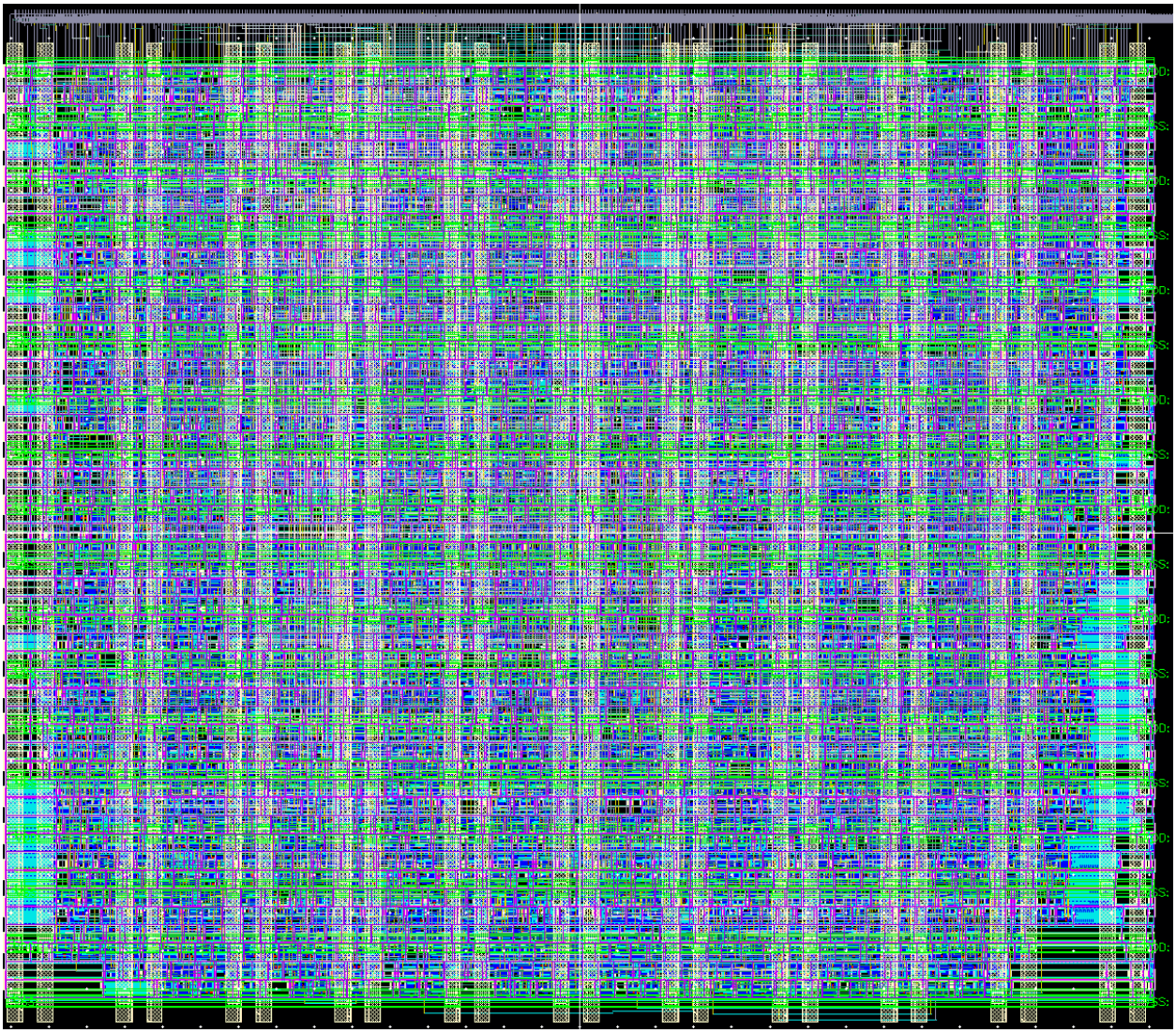

Figure 45: Synthesized High-performance Adder Layout

Table 12: Structural vs. Synthesized High-performance Adder Metrics

	Structural	Synthesized	Percent Improvement
Average Frequency (MHz)	515.8	406.6	-21.2%
Average Active Energy per Operation (pJ)	19.19	17.40	9.3%
EDP (pJ*ns)	37.21	42.80	-15.0%
Leakage Power (µW)	16.122	13.756	14.7%
Area (µm²)	11127	13258	-19.2%

Table 12 presents the metrics for the *structural* and *synthesized high-performance* adders, and the throughput warrants close examination. The *structural* CLA achieves 1.25 times the throughput of the *synthesized high-performance* adder. Figure 46 and Figure 47 present the *I paths* for the *structural* CLA and the *high-performance synthesized* adder, respectively. Notably, the logic depth of the *structural* design is significantly greater than that of the *synthesized* design. After removing the completion detection from the *TH24comp* down along with the buffers and register cells, the *structural* design has a logic depth of 14 MTNCL gates, whereas the *synthesized* design achieves a depth of only eight gates. In addition, the total delay through the *I path* of the *synthesized* adder is approximately *125 ps* less than that of the equivalent path in the *structural* design. Together, these two factors indicate that the *synthesized* design should outperform the *structural* design.

```
#-------------------------------------------------------------------------------------------
# Timing Point              Arc   Edge  Cell                      Fanout    Delay    Arrival
#                                                                            (ps)       (ps)
#-------------------------------------------------------------------------------------------
  ki                        ki    F     (arrival)                      1    0.100      0.100
  out_reg_compm_a_th22n/z   b->z  R     th22n_with_inv_LVT_X0P7N       2  211.300    211.400
  sleep_out_BufferLevel2_1  A->Y  R     BUF_X2B_A12TL                  1   55.300    266.700
  sleep_out_BufferLevel1_5  A->Y  R     BUF_X16B_A12TL                22   55.700    322.400
  sleep_out_BufferLevel1_1  A->Y  R     BUF_X16B_A12TL                29   59.000    381.400
  in_reg_compm_a_th22n/z    b->z  F     th22n_with_inv_LVT_X0P7N       2  207.800    589.200
  ko_BufferLevel2_5/Y       A->Y  F     BUF_X9B_A12TL                  5   66.200    655.400
  ko_BufferLevel2_3/Y       A->Y  F     BUF_X16B_A12TL                11   54.400    709.800
  ko_BufferLevel1_37/Y      A->Y  F     BUF_X16B_A12TL                28   61.000    770.800
  in_reg_G1_65_Gr0/z        s->z  R     th12m_LVT_HVT_X2N_skewed       4   55.200    826.000
  cla_rail0_1/z             c->z  R     thxor0m_LVT_HVT_X2N_skewed     8  111.100    937.100
  cla_p_g_generate_gen_1_0  c->z  R     th34w32m_LVT_HVT_X2N_skewed    1  106.500   1043.600
  cla_p_g_generate_gen_1_0  d->z  R     th54w32m_LVT_HVT_X2N_skewed    5  119.500   1163.100
  cla_p_g_generate_gen_2_0  d->z  R     th34w32m_LVT_HVT_X2N_skewed    1   99.000   1262.100
  cla_p_g_generate_gen_2_0  d->z  R     th54w32m_LVT_HVT_X2N_skewed    4  131.100   1393.200
  cla_CLA_2_0_c2_3/z        b->z  R     th34w32m_LVT_HVT_X2N_skewed    1   97.400   1490.600
  cla_CLA_2_0_c2_4/z        b->z  R     th22m_LVT_HVT_X2N_skewed       3   96.000   1586.600
  cla_CLA_1_2_c4_6/z        a->z  R     th14m_LVT_HVT_X2N_skewed       2   73.100   1659.700
  cla_CLA_1_2_c4_9/z        c->z  R     th23w2m_LVT_HVT_X2N_skewed     1   87.600   1747.300
  cla_CLA_1_2_c4_10/z       a->z  R     th33m_LVT_HVT_X2N_skewed       3   93.600   1840.900
  cla_CLA_0_12_c4_6/z       a->z  R     th14m_LVT_HVT_X2N_skewed       2   67.500   1908.400
  cla_CLA_0_12_c4_9/z       c->z  R     th23w2m_LVT_HVT_X2N_skewed     1   90.400   1998.800
  cla_CLA_0_12_c4_10/z      a->z  R     th33m_LVT_HVT_X2N_skewed       2   72.800   2071.600
  cla_sum_rail0             d->z  R     thxor0m_LVT_HVT_X2N_skewed     2   92.800   2164.400
  out_reg_compm26/z         a->z  R     th24compm_LVT_HVT_X2N_skewed   1   92.800   2257.200
  out_reg_compmG4_0_6/z     c->z  R     th44m_LVT_HVT_X2N_skewed       1   77.900   2335.100
  out_reg_compmG4_1_1/z     c->z  R     th44m_LVT_HVT_X2N_skewed       1   81.100   2416.200
  out_reg_compmG3F_2/z      b->z  R     th33m_LVT_HVT_X2N_skewed       1  116.300   2532.500
  out_compmth22n/a          a     R     th22n_with_inv_LVT_X0P7N       1    0.000   2532.500
#-------------------------------------------------------------------------------------------
```

Figure 46: Structural High-performance Adder Max DATA Cycle Path

```
#-------------------------------------------------------------------------------
# Timing Point          Arc   Edge  Cell                    Fanout    Delay    Arrival
#                                                                      (ps)       (ps)
#-------------------------------------------------------------------------------
  ki                    ki    F     (arrival)                    1    0.200      0.200
  block0_comp.th22/z    b->z  R     th22n_with_inv_LVT_X0P5N     1  193.900    194.100
  FE_O_sleep_out/Y      A->Y  R     BUF_X5B_A12TL               46  112.600    306.700
  comp1_inst_com/z      b->z  F     th22n_with_inv_LVT_X0P5N     1  185.400    492.100
  FE_OFC65_ko/Y         A->Y  F     BUF_X4B_A12TL               28   99.300    591.400
  FE_OFC0_ko/Y          A->Y  R     INV_X2P5B_A12TL             10  100.100    691.500
  FE_OFC19_ko/Y         A->Y  F     INV_X1B_A12TL               16  121.300    812.800
  FE_OFC39_ko/Y         A->Y  R     INV_X0P5B_A12TL              2   81.600    894.400
  FE_OFC48_ko/Y         A->Y  F     INV_X3P5B_A12TL             66  106.800   1001.200
  block0_reg_Gwi26].Gr  s->z  R     th12m_reg_LVT_HVT_X2N_skewed 4   58.200   1059.400
  block0_core_blpped_a  b->z  R     th22m_LVT_HVT_X2N_skewed     6   99.600   1159.000
  block0_core_FE1/z     c->z  R     th14m_LVT_HVT_X2N_skewed     1   67.700   1226.700
  FE_OFC253_blocblock0  A->Y  R     BUF_X0P7B_A12TL              7  100.000   1326.700
  block0_core_FE0/z     b->z  R     th14m_LVT_HVT_X2N_skewed     1   80.600   1407.300
  FE_OFC261_blocFE_RN_  A->Y  R     BUF_X0P7B_A12TL              1   49.900   1457.200
  block0_core_FE0/z     a->z  R     th23w2m_LVT_HVT_X1N_skewed   1   50.600   1507.800
  block0_core_FE0/z     c->z  R     th54w22m_LVT_HVT_X2N_skewed  1   87.200   1595.000
  FE_OFC177_blocblock0  A->Y  R     BUF_X1P7B_A12TL             20  134.400   1729.400
  block0_core_blpped_a  a->z  R     th12m_LVT_HVT_X1N_skewed     2   78.100   1807.500
  block0_core_blpped_a  a->z  R     th22m_LVT_HVT_X1N_skewed     2   78.600   1886.100
  block0_core_FE/z      d->z  R     thxor0m_LVT_HVT_X0P7N_skewed 1   93.000   1979.100
  FE_OFC223_blocblock0  A->Y  R     BUF_X0P7B_A12TL              1   64.200   2043.300
  block0_core_FE1/z     d->z  R     th24compm_LVT_HVT_X2N_skewed 1   86.200   2129.500
  block0_core_FE0/z     d->z  R     th44m_LVT_HVT_X2N_skewed     1   83.900   2213.400
  block0_core_g3        c->z  R     th44m_LVT_HVT_X2N_skewed     1  105.400   2318.800
  block0_rcomp.th22/a   a     R     th22n_with_inv_LVT_X0P5N     1    0.000   2318.800
#-------------------------------------------------------------------------------
```

Figure 47: Synthesized High-performance Adder Max DATA Cycle Path

Another counterintuitive observation is that the *structural* path seems too slow for the corresponding measured throughput. Even without considering the additional components of the critical path, the provided path corresponds to *395 MHz*, while the measured throughput of *516 MHz* would indicate a cycle time of *1,939 ps*. As discussed in Section 3.3.2, MTNCL achieves *average-case* performance due to its completion detection mechanism. To illustrate the performance range, Figure 48 provides the minimum delay for the same maximum delay path presented in Figure 46. Remarkably, the *best-case* delay is less than half the *worst-case* delay, with completion detection allowing the average delay to fall between these values.

```
#-------------------------------------------------------------------------------------
# Timing Point           Arc   Edge  Cell                    Fanout    Delay    Arrival
#                                                                       (ps)       (ps)
#-------------------------------------------------------------------------------------
  ki                     ki    F     (arrival)                    1    0.000      0.000
  out_reg_c_th22n/z      b->z  R     th22n_with_inv_LVT_X0P7N     2  206.700    206.700
  sleep_outrLevel2_1/Y   A->Y  R     BUF_X2B_A12TL                1   54.700    261.400
  sleep_outrLevel1_5/Y   A->Y  R     BUF_X16B_A12TL              22   55.200    316.600
  sleep_outrLevel1_1/Y   A->Y  R     BUF_X16B_A12TL              29   58.300    374.900
  in_reg_coth22n/z       b->z  F     th22n_with_inv_LVT_X0P7N     2  207.300    582.200
  ko_Buffer_5/Y          A->Y  F     BUF_X9B_A12TL                5   66.600    648.800
  ko_Buffer_4/Y          A->Y  F     BUF_X9B_A12TL                4   46.800    695.600
  ko_Buffer_39/Y         A->Y  F     BUF_X16B_A12TL              28   52.400    748.000
  in_reg_G10/z           s->z  R     th12m_LVT_HVT_X2N_skewed     4   37.600    785.600
  cla_rail0              c->z  R     thxor0m_LVT_HVT_X2N_skewed   7   89.300    874.900
  cla_sum_r3/z           c->z  R     thxor0m_LVT_HVT_X2N_skewed   2   75.700    950.600
  out_reg_c_Gs1_11/z     d->z  R     th24compm_LVT_HVT_X2N_skewed 1   61.000   1011.600
  out_reg_c_Gco_G4_0_2/z d->z  R     th44m_LVT_HVT_X2N_skewed     1   67.200   1078.800
  out_reg_c_Gco_G4_1_0/z c->z  R     th44m_LVT_HVT_X2N_skewed     1   84.200   1163.000
  out_reg_c_Gco_G3F_2/z  a->z  R     th33m_LVT_HVT_X2N_skewed     1   81.900   1244.900
  out_reg_c_th22n/a      a     R     th22n_with_inv_LVT_X0P7N     1    0.000   1244.900
#-------------------------------------------------------------------------------------
```

Figure 48: Structural High-performance Adder Min DATA Cycle Path

The explanation for the throughput discrepancy lies beyond the *I paths*. As laid out in Section 3.3.2, other path types, including the I/O paths, must be considered to ensure they do not bottleneck the design. Figure 49 and Figure 50 present the *worst-case* path from the *Ki* input, through the output completion detection and register, and concluding at one of the sum ports. The transition time for the nets is provided to demonstrate that the *synthesized* design was successfully buffered according to the *max_transition* constraint. The total buffering delay in the *structural* design is approximately *168 ps*, while the *synthesized* design exhibits about 2.3 times this buffering delay at *382 ps*. Furthermore, the slower transition time on the register *sleep* pin contributes *76 ps* to its propagation delay. Overall, this path is *274 ps* slower than the equivalent path in the *structural* adder, indicating the challenges of buffering a design for high performance using *max_transition* constraints.

```
#-------------------------------------------------------------------------------------
# Timing Point       Arc    Edge  Cell                  Fanout   Trans     Delay   Arrival
#                                                                (ps)      (ps)      (ps)
#-------------------------------------------------------------------------------------
  ki                 ki     R     (arrival)                  1  50.000     0.100     0.100
  out_rcompth22n/z   b->z   F     th22n_with_inv_LVT_X0P7N   2  50.000   186.900   187.000
  sleep_out_Buffer   A->Y   F     BUF_X2B_A12TL              1  24.000    54.800   241.800
  sleep_out_Buffer   A->Y   F     BUF_X16B_A12TL            22  13.500    55.500   297.300
  sleep_out_Buffer   A->Y   F     BUF_X16B_A12TL            29  20.500    58.100   355.400
  out_reg_G/z        s->z   R     th12m_LVT_HVT_X2N_skewed   2  21.400    32.700   388.100
  sum_dual_rail[2]           R     -                          2  13.600     0.000   388.100
#-------------------------------------------------------------------------------------
```

Figure 49: Structural High-performance Adder Ki-to-out Path

```
#-------------------------------------------------------------------------------------
# Timing Point      Arc   Edge  Cell                    Fanout   Trans     Delay   Arrival
#                                                                (ps)      (ps)      (ps)
#-------------------------------------------------------------------------------------
  ki                ki    R     (arrival)                    1  50.000     0.200     0.200
  block0_reth22/z   b->z  F     th22n_with_inv_LVT_X0P5N     1  50.000   171.700   171.900
  FE_sleep_out/Y    A->Y  F     BUF_X5B_A12TL               46  18.800   120.100   292.000
  FE_sleep_out/Y    A->Y  F     BUF_X3B_A12TL               26  77.100   135.300   427.300
  FE__sleep_out/Y   A->Y  F     BUF_X0P7B_A12TL              6  72.800   126.400   553.700
  out_reg[30]/z     s->z  R     th12m_reg_LVT_HVT_X2N_skewed 1  69.500   108.300   662.000
  sum_dual_rail[60]       R     -                            1  50.500     0.000   662.000
#-------------------------------------------------------------------------------------
```

Figure 50: Synthesized High-performance Adder Ki-to-out Path

This observation does not serve as an endorsement of the buffering scheme employed by the *structural* design, as this approach completely excludes the effects of parasitic routing resistance and capacitance. The custom buffering script targets cell input capacitance alone, whereas modern physical implementation tools leverage accurate RC models that account for routing, transition time, propagation delay, and crosstalk. The relative shortcomings of the buffering in the *synthesized* designs should not be interpreted as a condemnation of physical implementation tools; rather, they indicate that the tool was not employed correctly. As demonstrated in the next section of this chapter, the introduction of timing constraints enables the physical implementation utility to buffer the target design with significantly higher quality and accuracy than either of the buffering approaches applied in the two designs discussed in this section.

For 1-stage designs like the adders, the critical path is not composed of the two slowest stages. In the absence of a second *I path* through an adjacent stage, one or more alternate path types will form the bottleneck. Figure 51 and Figure 52 present the NULL cycle time for the *structural* and *synthesized* adders, respectively, beginning with a request for NULL from the output completion component and ending with the arrival of NULL at the same gate. These paths demonstrate that the NULL cycle time of the *synthesized high-performance* adder is *125 ps* slower than the *structural* design, providing additional context for the *structural* design's superior performance. Without a second *I path,* the overall circuit cycle time can be approximated by summing the DATA and NULL cycle times.

```
#-------------------------------------------------------------------------------------------
# Timing Point                Arc   Edge  Cell                   Fanout     Delay   Arrival
#                                                                          (ps)      (ps)
#-------------------------------------------------------------------------------------------
  ki                          ki    R     (arrival)                   1     0.000     0.000
  out_reg_compm_a_th22n/z     b->z  F     th22n_with_inv_LVT_X0P7N    2   182.800   182.800
  sleep_out_BufferLevel2_1/Y  A->Y  F     BUF_X2B_A12TL               1    54.700   237.500
  sleep_out_BufferLevel1_5/Y  A->Y  F     BUF_X16B_A12TL             22    55.100   292.600
  sleep_out_BufferLevel1_1/Y  A->Y  F     BUF_X16B_A12TL             29    57.000   349.600
  in_reg_compm_a_th22n/z      b->z  R     th22n_with_inv_LVT_X0P7N    2   241.100   590.700
  ko_BufferLevel2_5/Y         A->Y  R     BUF_X9B_A12TL               5    66.900   657.600
  ko_BufferLevel1_43/Y        A->Y  R     BUF_X16B_A12TL             21    57.500   715.100
  out_reg_compm_a_Gco_G3F_2/z s->z  F     th33m_LVT_HVT_X2N_skewed    1    45.500   760.600
  out_reg_compm_a_th22n/a     a     F     th22n_with_inv_LVT_X0P7N    1     0.000   760.600
#-------------------------------------------------------------------------------------------
```

Figure 51: Structural High-performance Adder Max NULL Cycle Path

```
#-------------------------------------------------------------------------------------
# Timing Point            Arc   Edge  Cell                      Fanout    Delay   Arrival
#                                                                          (ps)      (ps)
#-------------------------------------------------------------------------------------
  ki                      ki    R     (arrival)                      1    0.100     0.100
  block0_re_comp.th22/z   b->z  F     th22n_with_inv_LVT_X0P5N       1  171.600   171.700
  FE_sleep_out/Y          A->Y  F     BUF_X5B_A12TL                 46  101.200   272.900
  cost_comp_finish/z      b->z  R     th22n_with_inv_LVT_X0P5N       1  200.800   473.700
  FE_ko/Y                 A->Y  R     BUF_X4B_A12TL                 28  106.000   579.700
  FEko/Y                  A->Y  F     INV_X2B_A12TL                 10  111.100   690.800
  FE_ko/Y                 A->Y  R     INV_X4B_A12TL                 43  118.000   808.800
  block0g3868/z           s->z  F     th44m_LVT_HVT_X2N_skewed       1   76.600   885.400
  block0ull_comp.th22/a   a     F     th22n_with_inv_LVT_X0P5N       1    0.000   885.400
#-------------------------------------------------------------------------------------
```

Figure 52: Synthesized High-performance Adder Max NULL Cycle Path

An attentive reader may have noticed that the *400 ps* path delay difference presented thus far does not fully account for the additional *520 ps* of cycle delay observed in the *synthesized design*. A further portion can be attributed to the testbench setup for the designs. As explained in Section 2.5, it has been common practice to delay the assignment of NULL waves to MTNCL circuits during simulation to avoid the DATA handshaking timing race condition due to the unrealistic absence of delay at the input interface of the circuit. This approach was also necessary for most of the circuits used in this evaluation. A detailed analysis of these delay values for each design will be deferred until Section 5.5. However, while the CLA required a delay of only *100 ps*, the *synthesized high-performance* adder required a delay of *400 ps*. Together, these sources likely contribute to the total delay difference of *520 ps*.

Given the asynchronous nature of the circuits and the lack of comprehensive timing reports from physical implementation, it is difficult to precisely diagnose the exact cause(s) for the slowdown. Furthermore, the slowdown is almost certainly a multivariable issue. Nonetheless, the *Ki-to-output* path, NULL cycle time, and external testbench delay help explain the throughput difference observed between these circuits.

Although the *synthesized high-performance* adder has 19.2% more area than the *structural* CLA, it consumes 9.3% less active energy per operation and 14.7% less power while idle. Table 13 and Table 14 are provided to explain this initially counterintuitive trend by including a breakdown of each design in terms of gate type, gate size, and the approximate amount of switching for each category. The total switching metric is the product of the total area, size factor, and activity factor of the associated cell category. The size factor accounts for variations in switching energy due to cell drive strength, as even cells with the same footprint (e.g., *X0P5*, *X0P7*, and *X1* versions) consume more switching energy with higher drive strength. Additionally, the activity factor ensures that the higher switching activity of the *sleep* drivers is accounted for.

Table 13: Structural High-performance Adder Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	0	0	0.5	0.5	0
MTNCL X0P7	0	0	0.7	0.5	0
MTNCL X1	0	0	1	0.5	0
MTNCL X2	1459	10337	2	0.5	10337
BUF	59	777	1	1	777
INV	0	0	1	1	0
Total	1518	11113	0	0	11113

Table 14: Synthesized High-performance Adder Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	341	1622	0.5	0.5	406
MTNCL X0P7	454	2407	0.7	0.5	842
MTNCL X1	559	2786	1	0.5	1393
MTNCL X2	878	5986	2	0.5	5986
BUF	150	328	1	1	328
INV	46	115	1	1	115
Total	2428	13244	0	0	9070

A comparison of the tables provides several interesting observations. To achieve the 43% reduction in combinational logic depth previously discussed, the *synthesized* design required 53% more MTNCL gates. Despite the substantial increase in MTNCL cell count, the total MTNCL cell area of the *synthesized* adder is only 24% larger than the *structural* design. In contrast to the custom buffering script, which buffered the circuit with 59 large buffer cells, Innovus opted to use a larger number of smaller cells—about 3.3 times more. Using this fine-grained strategy, Innovus reduced the buffering area by 43%. Interestingly, although the *synthesized* design consists of 60% more cell instances and 19% greater cell area, it exhibits 18.3% less switching activity. This is a clear indicator that Innovus is using its cell area more effectively than the *structural* design, largely by downsizing the cells on less-critical paths.

These tables provide an approximate rather than a comprehensive explanation of the active energy difference between the two designs, due to several contributing factors. First, the size factor is an oversimplification—for example, the total dynamic energy does not precisely double when upsizing an *X0P5* gate to *X1*. Second, the activity factor is an approximation. Although the data pins (i.e., *a*, *b*, *c*, and *d*) have an activity factor of 0.5 due to the dual-rail encoding, the *sleep* transistors in each MTNCL gate have an activity factor of 1.0. Finally, the tables attribute all the

buffers and inverters to the *sleep* trees, though some are likely used to buffer data signals between MTNCL gates in the combinational logic and completion detection. Despite these approximations, the tables illustrate how the *synthesized* design achieves less active energy and leakage power despite its larger design area.

5.1.3.3 Synthesized vs. Optimized

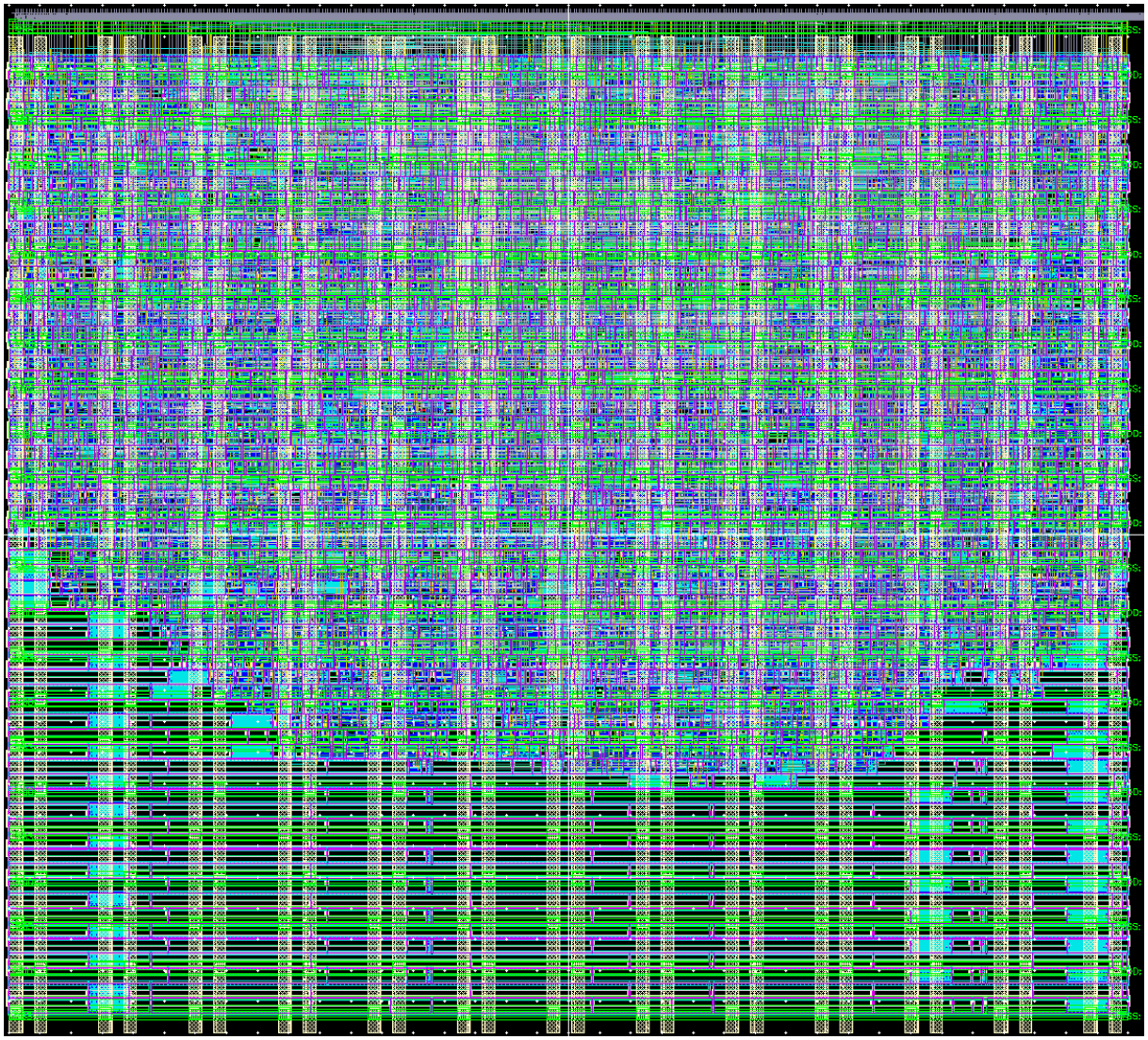

Figure 53: Optimized High-performance Adder Layout

Table 15: Synthesized vs. Optimized High-performance Adder Metrics

	Synthesized	Optimized	Percent Improvement
Average Frequency (MHz)	406.6	592.4	45.7%
Average Active Energy per Operation (pJ)	17.40	27.93	-60.5%
EDP (pJ*ns)	42.80	47.15	-10.1%
Leakage Power (µW)	13.756	35.571	-158.6%
Area (µm²)	13258	16847	-27.1%

Table 15 provides a comparison of the design metrics for the *synthesized* and *optimized high-performance* adders. The application of timing constraints during physical implementation increased throughput by 45.7%, elucidated by the *I paths* within the adders presented in Figure 47 and Figure 54. Innovus optimized the latter in response to the *I* constraint, which was set to *1,275 ps*. While the total delay through the MTNCL gates is similar between the two, the total buffering delays differ substantially: the *synthesized* adder encompasses *858 ps*, while the *optimized* adder incurs only *195 ps*—a 77% reduction. Instead of blindly buffering the circuit according to the max transition constraint, Innovus carefully selected and sized buffers and inverters to minimize path delay along the constrained path.

```
#-------------------------------------------------------------------------------
# Timing Point          Arc   Edge  Cell                    Fanout    Delay    Arrival
#                                                                      (ps)       (ps)
#-------------------------------------------------------------------------------
  ki                    ki    F     (arrival)                    1    0.200      0.200
  block0_comp.th22/z    b->z  R     th22n_with_inv_LVT_X0P5N     1  193.900    194.100
  FE_O_sleep_out/Y      A->Y  R     BUF_X5B_A12TL               46  112.600    306.700
  comp1_inst_com/z      b->z  F     th22n_with_inv_LVT_X0P5N     1  185.400    492.100
  FE_OFC65_ko/Y         A->Y  F     BUF_X4B_A12TL               28   99.300    591.400
  FE_OFC0_ko/Y          A->Y  R     INV_X2P5B_A12TL             10  100.100    691.500
  FE_OFC19_ko/Y         A->Y  F     INV_X1B_A12TL               16  121.300    812.800
  FE_OFC39_ko/Y         A->Y  R     INV_X0P5B_A12TL              2   81.600    894.400
  FE_OFC48_ko/Y         A->Y  F     INV_X3P5B_A12TL             66  106.800   1001.200
  block0_reg_Gwi26].Gr  s->z  R     th12m_reg_LVT_HVT_X2N_skewed 4   58.200   1059.400
  block0_core_blpped_a  b->z  R     th22m_LVT_HVT_X2N_skewed     6   99.600   1159.000
  block0_core_FE1/z     c->z  R     th14m_LVT_HVT_X2N_skewed     1   67.700   1226.700
  FE_OFC253_blocblock0  A->Y  R     BUF_X0P7B_A12TL              7  100.000   1326.700
  block0_core_FE0/z     b->z  R     th14m_LVT_HVT_X2N_skewed     1   80.600   1407.300
  FE_OFC261_blocFE_RN_  A->Y  R     BUF_X0P7B_A12TL              1   49.900   1457.200
  block0_core_FE0/z     a->z  R     th23w2m_LVT_HVT_X1N_skewed   1   50.600   1507.800
  block0_core_FE0/z     c->z  R     th54w22m_LVT_HVT_X2N_skewed  1   87.200   1595.000
  FE_OFC177_blocblock0  A->Y  R     BUF_X1P7B_A12TL             20  134.400   1729.400
  block0_core_blpped_a  a->z  R     th12m_LVT_HVT_X1N_skewed     2   78.100   1807.500
  block0_core_blpped_a  a->z  R     th22m_LVT_HVT_X1N_skewed     2   78.600   1886.100
  block0_core_FE/z      d->z  R     thxor0m_LVT_HVT_X0P7N_skewed 1   93.000   1979.100
  FE_OFC223_blocblock0  A->Y  R     BUF_X0P7B_A12TL              1   64.200   2043.300
  block0_core_FE1/z     d->z  R     th24compm_LVT_HVT_X2N_skewed 1   86.200   2129.500
  block0_core_FE0/z     d->z  R     th44m_LVT_HVT_X2N_skewed     1   83.900   2213.400
  block0_core_g3        c->z  R     th44m_LVT_HVT_X2N_skewed     1  105.400   2318.800
  block0_rcomp.th22/a   a     R     th22n_with_inv_LVT_X0P5N     1    0.000   2318.800
#-------------------------------------------------------------------------------
```

Figure 47: Synthesized High-performance Adder Max DATA Cycle Path

```
#-------------------------------------------------------------------------------------------
# Timing Point               Arc   Edge  Cell                      Fanout    Delay    Arrival
#                                                                              (ps)       (ps)
#-------------------------------------------------------------------------------------------
  ki                         ki    F     (arrival)                      1    0.200      0.200
  block0_reg_null_comp.th22  b->z  R     th22n_with_inv_LVT_X2N         3  139.200    139.400
  comp1_inst_comp_finish/z   b->z  F     th22n_with_inv_LVT_X2N         4  138.100    277.500
  FE_OCPC5485_FE_OFN0_ko/Y   A->Y  R     INV_X9B_A12TL                  3   30.500    308.000
  FE_OCPC5518_FE_OFN13_ko/Y  A->Y  F     INV_X16B_A12TL                14   32.200    340.200
  FE_OCPC5669_FE_OFN0_ko/Y   A->Y  R     INV_X16B_A12TL                14   34.100    374.300
  FE_PSC6005_FE_OCPN11ko/    A->Y  R     BUFH_X1M_A12TL                 3   56.600    430.900
  FE_OCPC5681_FE_OFN0_ko/Y   A->Y  F     INV_X2B_A12TL                  4   34.200    465.100
  block0_reg_Gwithreset[     s->z  R     th12m_reg_LVT_HVT_X2N_skewed   1   31.600    496.700
  FE_OCPC5705_block0_A_reg_  A->Y  R     BUF_X0P7B_A12TL                8  121.000    617.700
  block0_core_FE_RC_409_0_d  a->z  R     th12m_LVT_HVT_X2N_skewed       1   78.600    696.300
  block0_core_FE_RC_83_0/z   d->z  R     thxor0m_LVT_HVT_X2N_skewed     2   76.800    773.100
  block0_core_FE_RC_932_0/z  c->z  R     th13m_LVT_HVT_X1N_skewed       1   59.600    832.700
  block0_core_FE_RC_68_1/z   a->z  R     th54w22m_LVT_HVT_X2N_skewed    1   97.500    930.200
  block0_core_FE_RC_929_0/z  c->z  R     th23w2m_LVT_HVT_X1N_skewed     1   95.100   1025.300
  block0_core_FE_RC_925_0/z  c->z  R     th54w22m_LVT_HVT_X2N_skewed    1   87.800   1113.100
  FE_PSBC6013_block0_core    A->Y  R     BUF_X6M_A12TL                  6   48.000   1161.100
  FE_PSBC5909_FE_OFN370_b    A->Y  R     BUFH_X5M_A12TL                 6   28.500   1189.600
  block0_core_block0_wrap    a->z  R     th12m_LVT_HVT_X2N_skewed       4   62.600   1252.200
  block0_core_FE_RC_529_0/z  a->z  R     thxor0m_LVT_HVT_X2N_skewed     1   85.100   1337.300
  block0_core_FE_RC_174_0/z  b->z  R     th24compm_LVT_HVT_X2N_skewed   1  103.100   1440.400
  block0_core_g3871/z        a->z  R     th44m_LVT_HVT_X2N_skewed       1  121.900   1562.300
  block0_core_g3868/z        b->z  R     th44m_LVT_HVT_X2N_skewed       1  101.900   1664.200
  block0_reg_null_comp.th22  a     R     th22n_with_inv_LVT_X2N         1    0.000   1664.200
#-------------------------------------------------------------------------------------------
```

Figure 54: Optimized High-performance Adder Max DATA Cycle Path

To support discussion of additional design metrics, Table 14 and Table 16 present the composition, area, and switching activity of the *synthesized* and *optimized* adders, respectively. In response to stringent timing constraints, Innovus added an additional 1,308 buffer and inverter cells, increasing the associated instance count by 6.7× and the design area by 8×, which caused a 158% spike in leakage power.

Table 14: Synthesized High-performance Adder Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	341	1622	0.5	0.5	406
MTNCL X0P7	454	2407	0.7	0.5	842
MTNCL X1	559	2786	1	0.5	1393
MTNCL X2	878	5986	2	0.5	5986
BUF	150	328	1	1	328
INV	46	115	1	1	115
Total	2428	13244	0	0	9070

Table 16: Optimized High-performance Adder Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	287	1412	0.5	0.5	353
MTNCL X0P7	242	1271	0.7	0.5	445
MTNCL X1	831	4212	1	0.5	2106
MTNCL X2	872	5946	2	0.5	5946
BUF	1230	3152	1	1	3152
INV	274	840	1	1	840
Total	3736	16832	0	0	12841

Overall, the design area increased by *3,588 µm²*, a 27% increase from the *synthesized* design. In addition to buffering, a subset of the MTNCL gates were upsized, increasing the MTNCL switching activity by a marginal 2.5%. The total switching activity increased by 41% in response to the timing constraints, and 88% of this increase can be attributed to the buffering. Together, the buffering and upsizing increased the average active energy per operation by 61%, from *17.4 pJ* to *27.9 pJ*. After accounting for the improved performance and degraded active energy, the EDP of the *optimized high-performance* adder worsened by 10%. Despite the power and area penalties, the application of timing constraints increased throughput by 46% and yielded a

significantly more reliable design, making the *optimized* adder superior for *high-performance* applications.

5.1.3.4 Summary

Table 17: Structural vs. Optimized High-performance Adder Metrics

	Structural	Optimized	Percent Improvement
Average Frequency (MHz)	515.8	592.4	14.9%
Average Active Energy per Operation (pJ)	19.19	27.93	-45.5%
EDP (pJ*ns)	37.21	47.15	-26.7%
Leakage Power (µW)	16.122	35.571	-120.6%
Area (µm²)	11127	16847	-51.4%

Table 17 offers a comparison of the *structural* CLA and the *optimized* adder. At first glance, a 14.9% improvement in throughput seems underwhelming given the application of the synthesis flow and timing constraints in the *optimized* design. Additionally, the *I paths* for the *structural* and *optimized* adders, presented in Figure 46 and Figure 54, indicate that the *optimized* design should far outperform the *structural* CLA. However, the measured performance of the *structural* design is misleading. Due to the lack of logic adjacent to the single stage, the handshaking on the *Ko* and *Ki* is very fast, translating to a cycle time derived almost exclusively from a single *I path* delay. This rapid handshaking at the input interface, however, exposes the design to the DATA handshaking race condition discussed in Section 2.5. Figure 55 and Figure 56 provide the two paths involved in the race condition. Since *Ko* switches before the register latches its DATA, the design risks deadlocking if the request for NULL is serviced too quickly.

```
#-------------------------------------------------------------------------------------------
# Timing Point              Arc   Edge  Cell                      Fanout    Delay    Arrival
#                                                                            (ps)       (ps)
#-------------------------------------------------------------------------------------------
  ki                        ki    F     (arrival)                      1    0.100      0.100
  out_reg_compm_a_th22n/z   b->z  R     th22n_with_inv_LVT_X0P7N       2  211.300    211.400
  sleep_out_BufferLevel2_1  A->Y  R     BUF_X2B_A12TL                  1   55.300    266.700
  sleep_out_BufferLevel1_5  A->Y  R     BUF_X16B_A12TL                22   55.700    322.400
  sleep_out_BufferLevel1_1  A->Y  R     BUF_X16B_A12TL                29   59.000    381.400
  in_reg_compm_a_th22n/z    b->z  F     th22n_with_inv_LVT_X0P7N       2  207.800    589.200
  ko_BufferLevel2_5/Y       A->Y  F     BUF_X9B_A12TL                  5   66.200    655.400
  ko_BufferLevel2_3/Y       A->Y  F     BUF_X16B_A12TL                11   54.400    709.800
  ko_BufferLevel1_37/Y      A->Y  F     BUF_X16B_A12TL                28   61.000    770.800
  in_reg_G1_65_Gr0/z        s->z  R     th12m_LVT_HVT_X2N_skewed       4   55.200    826.000
  cla_rail0_1/z             c->z  R     thxor0m_LVT_HVT_X2N_skewed     8  111.100    937.100
  cla_p_g_generate_gen_1_0  c->z  R     th34w32m_LVT_HVT_X2N_skewed    1  106.500   1043.600
  cla_p_g_generate_gen_1_0  d->z  R     th54w32m_LVT_HVT_X2N_skewed    5  119.500   1163.100
  cla_p_g_generate_gen_2_0  d->z  R     th34w32m_LVT_HVT_X2N_skewed    1   99.000   1262.100
  cla_p_g_generate_gen_2_0  d->z  R     th54w32m_LVT_HVT_X2N_skewed    4  131.100   1393.200
  cla_CLA_2_0_c2_3/z        b->z  R     th34w32m_LVT_HVT_X2N_skewed    1   97.400   1490.600
  cla_CLA_2_0_c2_4/z        b->z  R     th22m_LVT_HVT_X2N_skewed       3   96.000   1586.600
  cla_CLA_1_2_c4_6/z        a->z  R     th14m_LVT_HVT_X2N_skewed       2   73.100   1659.700
  cla_CLA_1_2_c4_9/z        c->z  R     th23w2m_LVT_HVT_X2N_skewed     1   87.600   1747.300
  cla_CLA_1_2_c4_10/z       a->z  R     th33m_LVT_HVT_X2N_skewed       3   93.600   1840.900
  cla_CLA_0_12_c4_6/z       a->z  R     th14m_LVT_HVT_X2N_skewed       2   67.500   1908.400
  cla_CLA_0_12_c4_9/z       c->z  R     th23w2m_LVT_HVT_X2N_skewed     1   90.400   1998.800
  cla_CLA_0_12_c4_10/z      a->z  R     th33m_LVT_HVT_X2N_skewed       2   72.800   2071.600
  cla_sum_rail0             d->z  R     thxor0m_LVT_HVT_X2N_skewed     2   92.800   2164.400
  out_reg_compm26/z         a->z  R     th24compm_LVT_HVT_X2N_skewed   1   92.800   2257.200
  out_reg_compmG4_0_6/z     c->z  R     th44m_LVT_HVT_X2N_skewed       1   77.900   2335.100
  out_reg_compmG4_1_1/z     c->z  R     th44m_LVT_HVT_X2N_skewed       1   81.100   2416.200
  out_reg_compmG3F_2/z      b->z  R     th33m_LVT_HVT_X2N_skewed       1  116.300   2532.500
  out_compmth22n/a          a     R     th22n_with_inv_LVT_X0P7N       1    0.000   2532.500
#-------------------------------------------------------------------------------------------
```

Figure 46: Structural High-performance Adder Max DATA Cycle Path

```
#-------------------------------------------------------------------------------------
# Timing Point               Arc   Edge  Cell                      Fanout    Delay    Arrival
#                                                                              (ps)       (ps)
#-------------------------------------------------------------------------------------
  ki                         ki    F     (arrival)                      1    0.200      0.200
  block0_reg_null_comp.th22  b->z  R     th22n_with_inv_LVT_X2N         3  139.200    139.400
  comp1_inst_comp_finish/z   b->z  F     th22n_with_inv_LVT_X2N         4  138.100    277.500
  FE_OCPC5485_FE_OFN0_ko/Y   A->Y  R     INV_X9B_A12TL                  3   30.500    308.000
  FE_OCPC5518_FE_OFN13_ko/Y  A->Y  F     INV_X16B_A12TL                14   32.200    340.200
  FE_OCPC5669_FE_OFN0_ko/Y   A->Y  R     INV_X16B_A12TL                14   34.100    374.300
  FE_PSC6005_FE_OCPN11ko/    A->Y  R     BUFH_X1M_A12TL                 3   56.600    430.900
  FE_OCPC5681_FE_OFN0_ko/Y   A->Y  F     INV_X2B_A12TL                  4   34.200    465.100
  block0_reg_Gwithreset[     s->z  R     th12m_reg_LVT_HVT_X2N_skewed   1   31.600    496.700
  FE_OCPC5705_block0_A_reg_  A->Y  R     BUF_X0P7B_A12TL                8  121.000    617.700
  block0_core_FE_RC_409_0_d  a->z  R     th12m_LVT_HVT_X2N_skewed       1   78.600    696.300
  block0_core_FE_RC_83_0/z   d->z  R     thxor0m_LVT_HVT_X2N_skewed     2   76.800    773.100
  block0_core_FE_RC_932_0/z  c->z  R     th13m_LVT_HVT_X1N_skewed       1   59.600    832.700
  block0_core_FE_RC_68_1/z   a->z  R     th54w22m_LVT_HVT_X2N_skewed    1   97.500    930.200
  block0_core_FE_RC_929_0/z  c->z  R     th23w2m_LVT_HVT_X1N_skewed     1   95.100   1025.300
  block0_core_FE_RC_925_0/z  c->z  R     th54w22m_LVT_HVT_X2N_skewed    1   87.800   1113.100
  FE_PSBC6013_block0_core    A->Y  R     BUF_X6M_A12TL                  6   48.000   1161.100
  FE_PSBC5909_FE_OFN370_b    A->Y  R     BUFH_X5M_A12TL                 6   28.500   1189.600
  block0_core_block0_wrap    a->z  R     th12m_LVT_HVT_X2N_skewed       4   62.600   1252.200
  block0_core_FE_RC_529_0/z  a->z  R     thxor0m_LVT_HVT_X2N_skewed     1   85.100   1337.300
  block0_core_FE_RC_174_0/z  b->z  R     th24compm_LVT_HVT_X2N_skewed   1  103.100   1440.400
  block0_core_g3871/z        a->z  R     th44m_LVT_HVT_X2N_skewed       1  121.900   1562.300
  block0_core_g3868/z        b->z  R     th44m_LVT_HVT_X2N_skewed       1  101.900   1664.200
  block0_reg_null_comp.th22  a     R     th22n_with_inv_LVT_X2N         1    0.000   1664.200
#-------------------------------------------------------------------------------------
```

Figure 54: Optimized High-performance Adder Max DATA Cycle Path

```
#-------------------------------------------------------------------------------------
# Timing Point          Arc   Edge  Cell                       Fanout    Delay   Arrival
#                                                                          (ps)      (ps)
#-------------------------------------------------------------------------------------
  in_reg_compm_a_th22n b     R     (arrival)                     29       -     0.000
  in_reg_compm_a_th22n b->z  F     th22n_with_inv_LVT_X0P7N       2  205.800   205.800
  ko                   ko    F     -                              2    0.000   205.800
#-------------------------------------------------------------------------------------
```

Figure 55: Structural High-performance Adder DATA Handshaking Hazard Path 1

```
#-------------------------------------------------------------------------------------
# Timing Point           Arc   Edge  Cell                       Fanout    Delay   Arrival
#                                                                          (ps)      (ps)
#-------------------------------------------------------------------------------------
  in_reg_compm_a_th22n  b     R     (arrival)                     29       -   381.400
  in_reg_compm_a_th22n  b->z  F     th22n_with_inv_LVT_X0P7N       2  207.800   589.200
  ko_BufferLevel2_5/Y   A->Y  F     BUF_X9B_A12TL                  5   66.300   655.500
  ko_BufferLevel2_3/Y   A->Y  F     BUF_X16B_A12TL                11   54.400   709.900
  ko_BufferLevel1_37/Y  A->Y  F     BUF_X16B_A12TL                28   60.700   770.600
  in_reg_G1_68_Gr1/z    s->z  R     th12m_LVT_HVT_X2N_skewed       -   57.100   827.700
#-------------------------------------------------------------------------------------
```

Figure 56: Structural High-performance Adder DATA Handshaking Hazard Path 2

The *optimized* adder, in contrast, incorporates additional delay in the handshaking to protect against the same race condition, as shown in Figure 57 and Figure 58. Additional delays, such as the one applied to *Ko* in Figure 57, help explain why the *optimized* adder has a relatively modest throughput compared to its presented *I path* delay. Notably, the *structural* design's competitive performance will vanish when it is integrated into a circuit with adjacent combinational logic. Assuming the adjacent stages have the same DATA delay, using the delay value from Figure 46 and Equation 2, the *structural* CLA's performance would likely drop to approximately *197 MHz*.

```
#-------------------------------------------------------------------------------------
# Timing Point            Arc   Edge  Cell                    Fanout     Delay   Arrival
#                                                                          (ps)      (ps)
#-------------------------------------------------------------------------------------
  comp1_inst_comp_finish  b     F     (arrival)                    3        -   270.100
  comp1_inst_comp_finish  b->z  R     th22n_with_inv_LVT_X2N       4  142.900   413.000
  FE_PSC5902_FE_OFN0_ko/  A->Y  R     BUFH_X1P7M_A12TL             2   46.500   459.500
  FE_OCPC5481_FE_OFN0_ko  A->Y  F     INV_X4B_A12TL                7   29.200   488.700
  FE_OCPC5491_FE_OFN0_ko  A->Y  R     INV_X2B_A12TL                5   49.500   538.200
  FE_PHC6109_ko/Y         A->Y  R     BUFH_X3M_A12TL               3   40.700   578.900
  FE_PHC6115_ko/Y         A->Y  R     BUF_X0P7B_A12TL              1   54.300   633.200
  FE_PHC6112_ko/Y         A->Y  R     BUF_X0P7B_A12TL              1   56.300   689.500
  ko                      ko    R     -                            1    0.000   689.500
#-------------------------------------------------------------------------------------
```

Figure 57: Optimized High-performance Adder DATA Handshaking Safeguard Path 1

```
#-------------------------------------------------------------------------------------
# Timing Point          Arc   Edge  Cell                    Fanout    Delay   Arrival
#                                                                      (ps)      (ps)
#-------------------------------------------------------------------------------------
  comp1_inst_compsh/b   b     F     (arrival)                    3        -   270.100
  comp1_inst_compsh/z   b->z  R     th22n_with_inv_LVT_X2N       4  143.100   413.200
  FE_OCPC5485_FE_ko/Y   A->Y  F     INV_X9B_A12TL                3   33.300   446.500
  FE_OCPC5518_FE__ko/Y  A->Y  R     INV_X16B_A12TL              14   36.400   482.900
  FE_OCPC5669_FE_ko/Y   A->Y  F     INV_X16B_A12TL              14   33.700   516.600
  FE_PSC6005_FE_O79_ko  A->Y  F     BUFH_X1M_A12TL               3   53.000   569.600
  FE_OCPC5681_FE_ko/Y   A->Y  R     INV_X2B_A12TL                4   31.100   600.700
  block0_reg_Gwitt[77]  s->z  F     th12m_reg_LVT_HVT_X2N_skewed -   33.300   634.000
#-------------------------------------------------------------------------------------
```

Figure 58: Optimized High-performance Adder DATA Handshaking Safeguard Path 2

As expected, the *optimized* design can now achieve higher performance because the additional delay added to the handshaking would be effectively hidden by the adjacent stage's *I path* delay. Using the delay from Figure 54 and Equation 2, the design would then achieve a throughput in the ballpark of *300 MHz*, which is 1.5 times greater than the *structural* design. Despite the substantial active energy, leakage power, and area penalties associated with the *optimized* design, this more accurate throughput analysis affirms that the *optimized* design is superior to the *structural* design due to its significantly higher performance and reliability—both imparted through the application of timing constraints.

5.2 Montgomery Modular Multipliers

5.2.1 Architecture

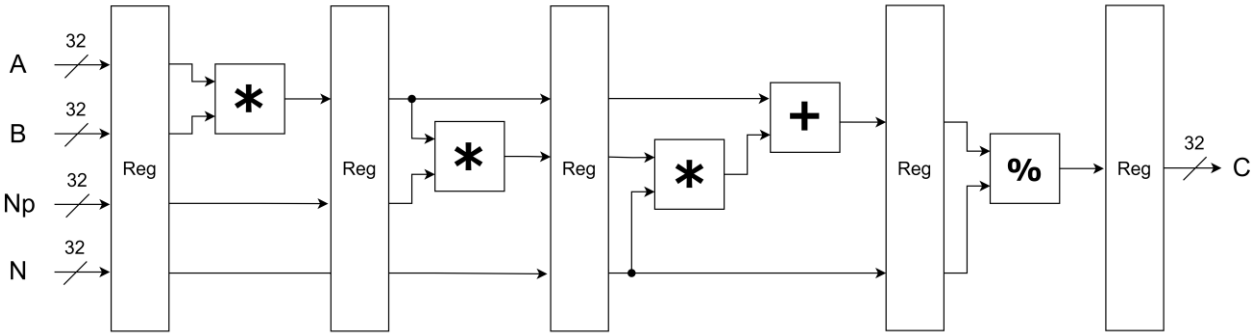

Figure 59: 32×32 Montgomery Modular Multiplier Circuit Architecture

Despite the term 'multiplier' being in the name, Montgomery modular multipliers are more complex and significantly larger than standard multiplier architectures like Baugh-Wooley, making them a more attractive test vehicle for the synthesis flow and timing constraints developed in this work. Figure 59 presents the structure of a 32×32 Montgomery modular multiplier. The *A* and *B* inputs are the values to be multiplied, *N* is the modulus, and *Np* is a value derived from the parameters used in the Montgomery reduction. For further information on

Montgomery reduction, the selection of N and Np, and how Montgomery modular multiplication works mathematically, please see [4].

The circuit is composed of three 32×32 unsigned multipliers, one 64-bit adder, and a single 32-bit modulo circuit, which are organized into four pipeline stages that leverage natural separations in the logic. The first two stages are nearly identical, each consisting of a single 32×32 multiplier. The third stage includes the third multiplier and the 64-bit adder, while the fourth and final stage features the modulo circuit, implemented with a comparator, subtractor, and multiplexer. The circuit output, C, is then driven by the output of the fifth pipeline register.

5.2.2 Low Power

5.2.2.1 Implementation

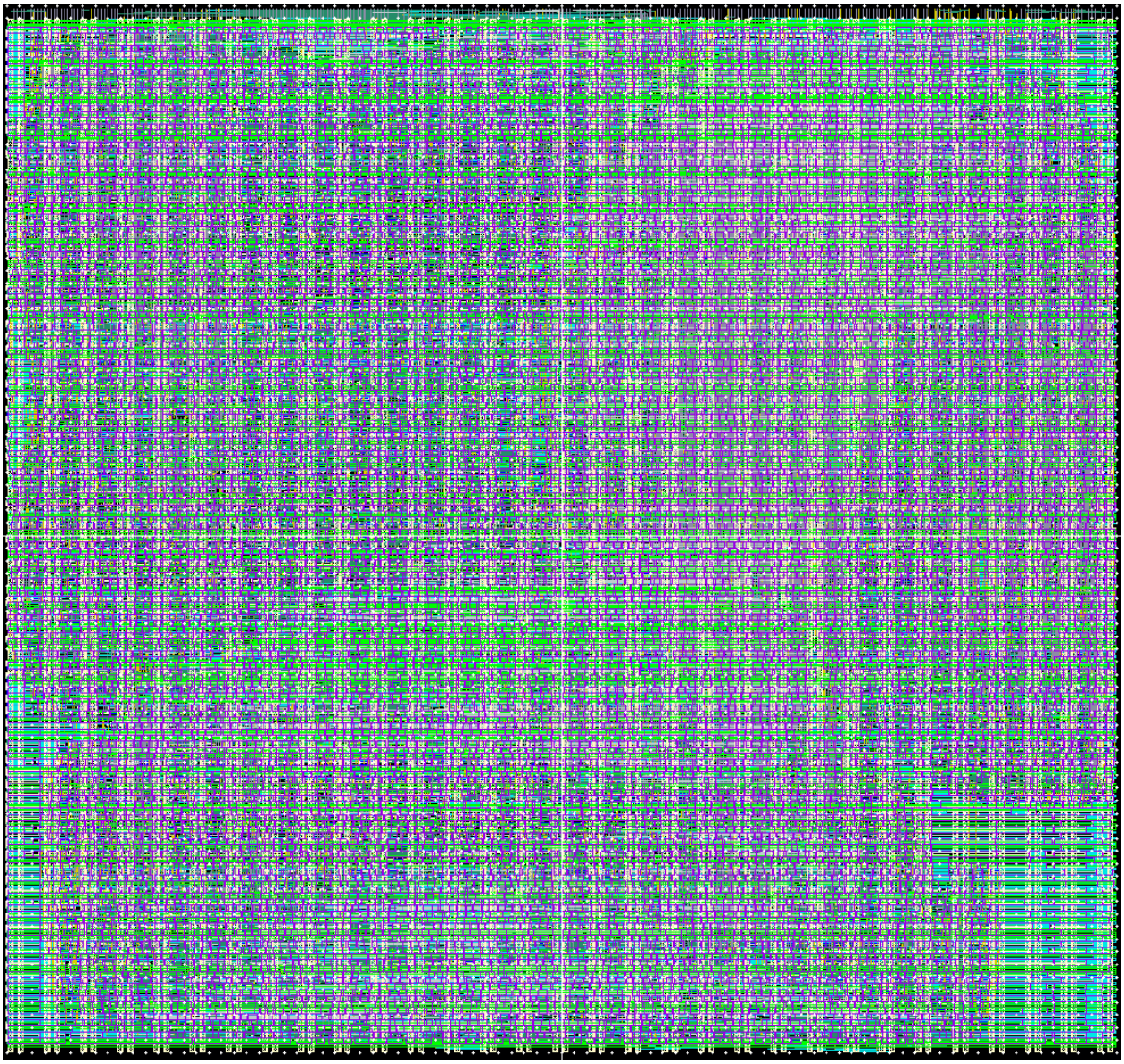

Figure 60: Structural Low-power Multiplier Layout

Pipelining only serves to enhance performance, so all the *low-power* Montgomery modular multipliers are designed as a single stage. Their high-level architecture is identical to Figure 59, apart from the removal of the three intermediate pipelining registers and adjacent completion detection units.

For *structural* design, in the absence of logic synthesis, architecture selection for the combinational components is often limited to pre-existing circuits. For this reason, the three multipliers were implemented as array multipliers, and both the adder and subtractor in the modulo circuit were implemented using the ripple-carry approach, given its availability and *low-power* nature. All four completion detection units were implemented structurally using balanced *AND trees*. Similar to the *low-power* RCA, all gate sizes were minimized prior to buffering with the custom script to reduce power consumption.

In contrast, the behavioral RTL provided to Genus for generating the *synthesized* implementation was relatively simple, comprising less than 100 lines of Verilog. Genus was then able to select from among many potential architectures for the arithmetic subcomponents, defaulting to *very high-performance* architectures and performing intense optimizations thereafter. Although Genus initially reported selection of *very_fast booth* multipliers, it is impossible to report—and thus identify—the architecture of the final multipliers following optimization. A deliberately slow *20 ns* clock was provided to Genus to facilitate maximal power optimization, resulting in *9 ns* of positive slack during single-rail synthesis.

The *optimized* design was built upon the *synthesized* equivalent by applying the timing constraints developed in this work. A very-loose value of *25 ns* was provided to Innovus for constraining the *I path* to prioritize power optimization, resulting in *3.7 ns* of positive slack. To this end, Innovus used minimum-sized gates ubiquitously throughout the design.

5.2.2.2 Structural vs. Synthesized

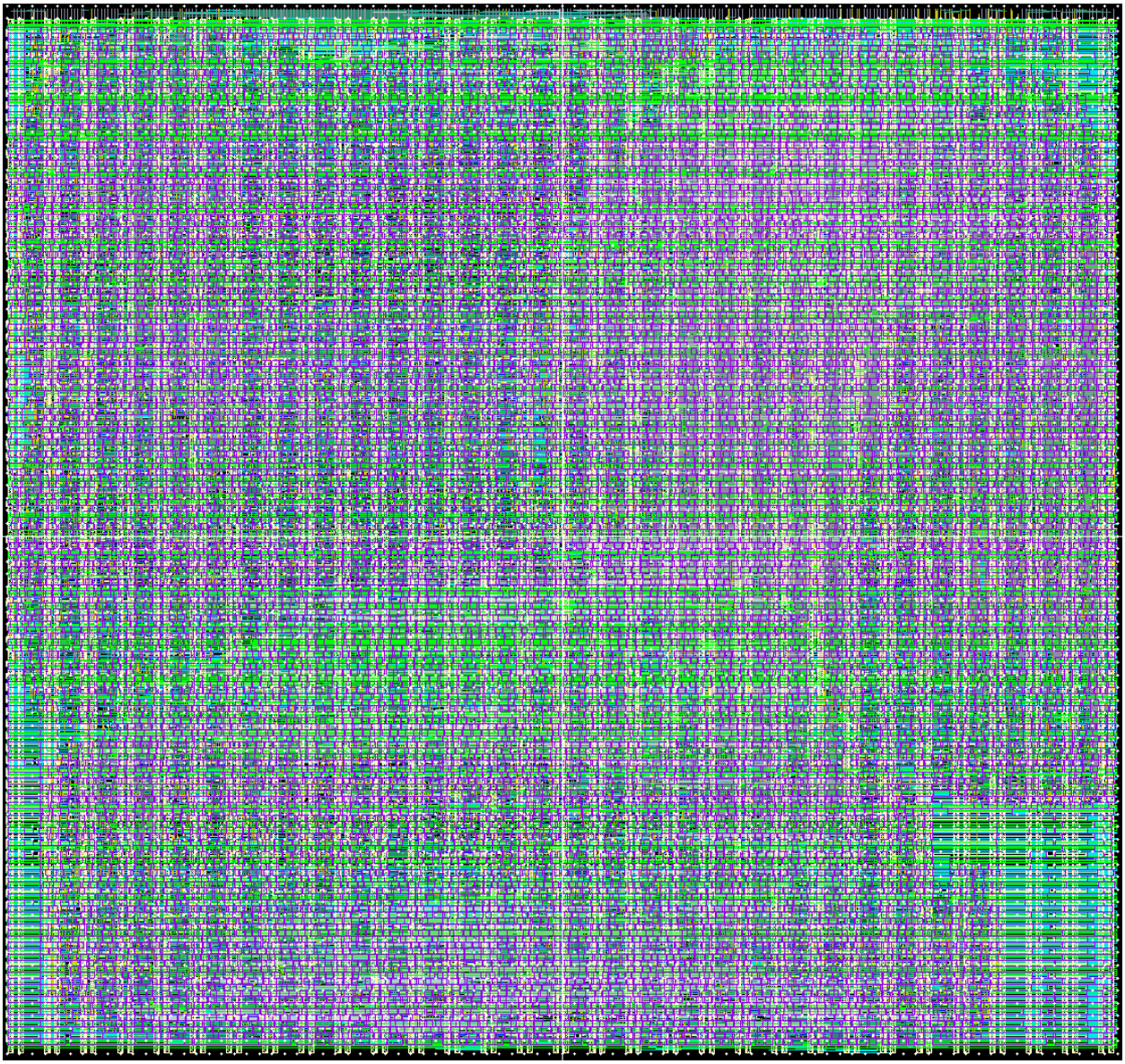

Figure 61: Synthesized Low-power Multiplier Layout

Table 18: Structural vs. Synthesized Low-power Multiplier Metrics

	Structural	Synthesized	Percent Improvement
Average Frequency (MHz)	68.2	103.2	51.4%
Average Active Energy per Operation (pJ)	126.50	89.17	29.5%
EDP (pJ*ns)	1854.92	863.84	53.4%
Leakage Power (µW)	108.991	52.840	51.5%
Area (µm²)	123318	85361	30.8%

Table 18 showcases the measured metrics for the *structural* and *synthesized low-power* Montgomery modular multipliers. This circuit type is a better example of the substantial improvements achievable through the transition from structural design to a high-quality synthesis flow, i.e., significant improvement across nearly all design metrics.

One of the most notable improvements is the throughput increase from *68.2 MHz* to *103.2 MHz*, representing a 51.4% increase. The majority of this speedup can be attributed to a reduction in logic depth. The *I path* of the *structural* design traverses through a total of 244 gates, while the *synthesized* design's *I path* consists of only 170 gates. Given the three multipliers, most of the circuit logic performs partial product generation and subsequent summation.

A few things can be inferred about the *synthesized* circuit architecture from the gate naming along the path. It appears that Genus decomposed the three multipliers and performed some type of merging optimization. The path includes 9 gates with *csa_tree* in their name, followed by 52 gates composing a Wallace tree adder, 62 gates labeled with *gte* implementing the modulo's comparator, and 33 gates with *SUB_UNS_OP* making the modulo's subtractor. The length and

repetitive gate pattern of the comparator and subtractor, close to 64 and 32 gates, respectively, indicate that Genus decided against unrolling the logic. Furthermore, the selection of *TH23* cells in the subtractor indicates a ripple-carry structure. Conversely, the *I path* for the *structural* design lacks any of these logical optimizations, with the boundary of each logic block clearly distinguishable.

The overall improvement can be further explained by the gate types used. The *structural* design used a total of 14 distinct gates, including 7 of the 27 fundamental threshold gate types. In contrast, Genus and Innovus used 63 distinct gate types and 18 of the 27 fundamental threshold gate types in the *synthesized* design, meaning that the EDA tools leveraged a wider variety of cells to fine tune the circuit. In place of the full gate lists, summaries of the cell categories and their associated switching activity have been provided in Table 19 and Table 20. The *synthesized* design has just 13,689 gates, a 30% reduction compared to the *structural* multiplier. Despite the *structural* design being composed exclusively of minimum-size MTNCL gates, projections suggest that it has 36.5% more switching activity in these gates. Notably, the custom buffering script increased the area and switching activity of the *sleep* drivers by 1.46×. This increase in switching activity accounts for the 29.5% reduction in active energy per operation that the *synthesized* design achieves. Combined with the 51.4% improvement in throughput, the energy savings lead to a nearly halved EDP for the *synthesized* design. The larger design area and significant increase in *sleep* driver area result in the *structural* design consuming more than double the static power. Overall, the *synthesized* design is far superior to the *structural* design for *low-power* applications, as substantial power savings can be attained during both operation and idle states. Additionally, supply voltage scaling could also be employed to trade off the performance gains for even lower power consumption.

Table 19: Structural Low-power Multiplier Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	18930	119134	0.5	0.5	29784
MTNCL X0P7	0	0	0.7	0.5	0
MTNCL X1	0	0	1	0.5	0
MTNCL X2	0	0	2	0.5	0
BUF	571	4170	1	1	4170
INV	0	0	1	1	0
Total	19501	123304	0	0	33953

Table 20: Synthesized Low-power Multiplier Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	12514	81078	0.5	0.5	20269
MTNCL X0P7	156	1026	0.7	0.5	359
MTNCL X1	132	767	1	0.5	384
MTNCL X2	104	798	2	0.5	798
BUF	729	1606	1	1	1606
INV	54	83	1	1	83
Total	13689	85357	0	0	23499

5.2.2.3 Synthesized vs. Optimized

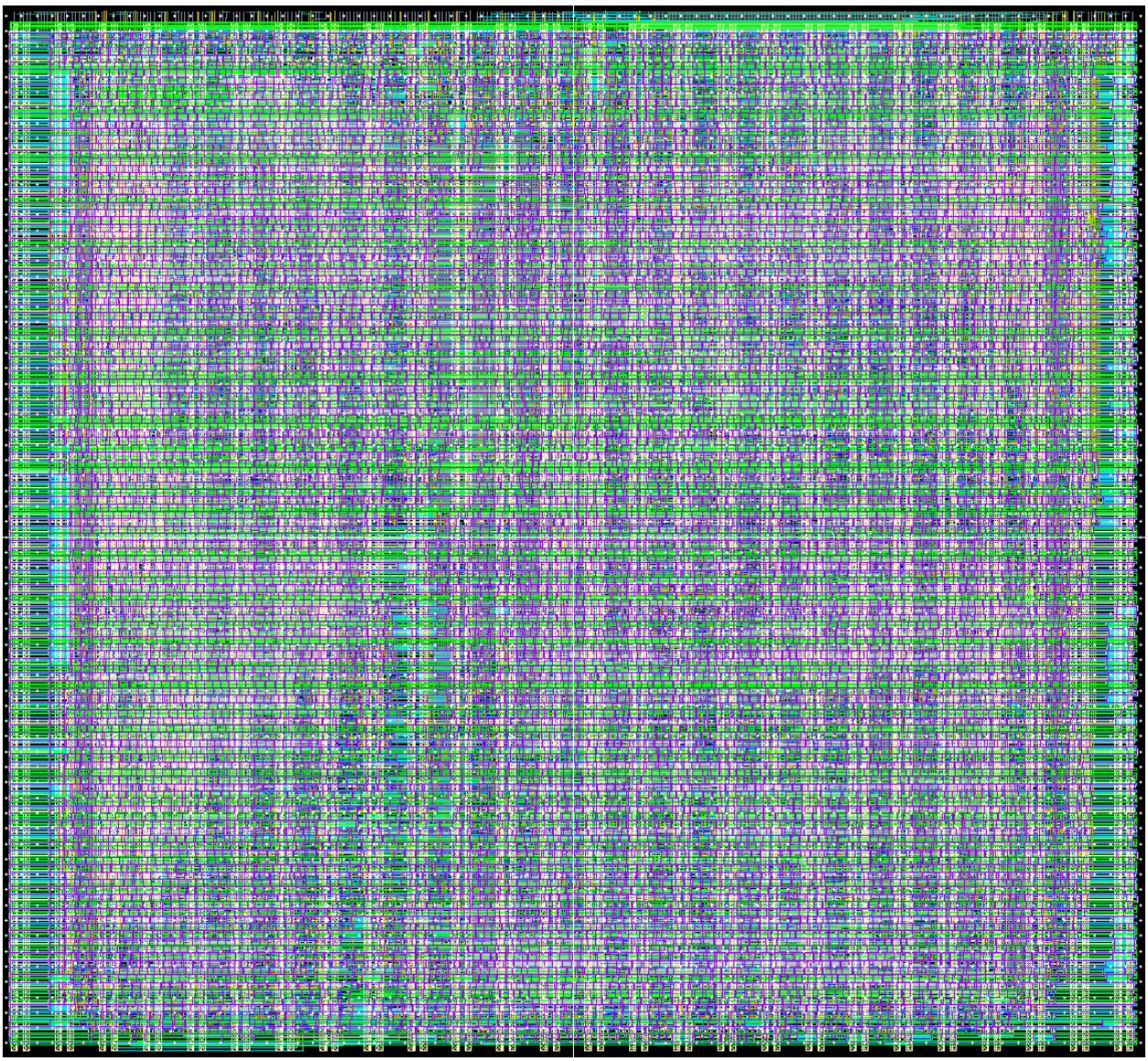

Figure 62: Optimized Low-power Multiplier Layout

Table 21: Synthesized vs. Optimized Low-power Multplier Metrics

	Synthesized	Optimized	Percent Improvement
Average Frequency (MHz)	103.2	108.8	5.4%
Average Active Energy per Operation (pJ)	89.17	90.35	-1.3%
EDP (pJ*ns)	863.84	830.62	3.8%
Leakage Power (µW)	52.840	62.189	-17.7%
Area (µm²)	85361	86828	-1.7%

Table 21 above allows for a comparison between the *synthesized* and *optimized low-power* Montgomery modular multipliers. The application of timing constraints during physical implementation had minor effects on most design metrics. The 5.4% throughput increase of the *optimized* design can be attributed to a 6.7% reduction in the *I path* delay. Table 20 has been included again below for reference alongside the corresponding Table 22 for the *optimized* design. The increase in switching activity is largely attributable to the buffers and inverters. Since Innovus was not optimizing the circuit for *high-performance*, these drivers were likely inserted to address hold timing requirements related to the reliability timing constraints. The total number of drivers increased from 783 to 1,359—a 73% increase. The increase in these high-leakage cells explains the 17.7% increase in static power dissipation and the 1.7% increase in design area. Although both active energy and static power degraded, reliability is paramount and outweighs other measured design metrics. For this reason, the *optimized* multiplier is better suited for *low-power* applications than the *synthesized* version.

Table 20: Synthesized Low-power Multiplier Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	12514	81078	0.5	0.5	20269
MTNCL X0P7	156	1026	0.7	0.5	359
MTNCL X1	132	767	1	0.5	384
MTNCL X2	104	798	2	0.5	798
BUF	729	1606	1	1	1606
INV	54	83	1	1	83
Total	13689	85357	0	0	23499

Table 22: Optimized Low-power Multiplier Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	12555	81334	0.5	0.5	20333
MTNCL X0P7	71	448	0.7	0.5	157
MTNCL X1	163	1096	1	0.5	548
MTNCL X2	117	800	2	0.5	800
BUF	756	2080	1	1	2080
INV	603	1067	1	1	1067
Total	14265	86824	0	0	24985

5.2.2.4 Summary

Table 23: Structural vs. Optimized Low-power Multiplier Metrics

	Structural	Optimized	Percent Improvement
Average Frequency (MHz)	68.2	108.8	59.5%
Average Active Energy per Operation (pJ)	126.50	90.35	28.6%
EDP (pJ*ns)	1854.92	830.62	55.2%
Leakage Power (µW)	108.991	62.189	42.9%
Area (µm²)	123318	86828	29.6%

Table 23 summarizes the design metric improvements gained through the application of a quality synthesis flow and timing constraints during the physical implementation of a l*ow-powe*r Montgomery modular multiplier. These improvements are similar to those discussed in Section 5.2.2.2, constituting substantial enhancements to throughput, active energy, leakage power, and area. Moreover, the design can be trusted to operate robustly across the analyzed PVT corners.

5.2.3 High Performance

5.2.3.1 Implementation

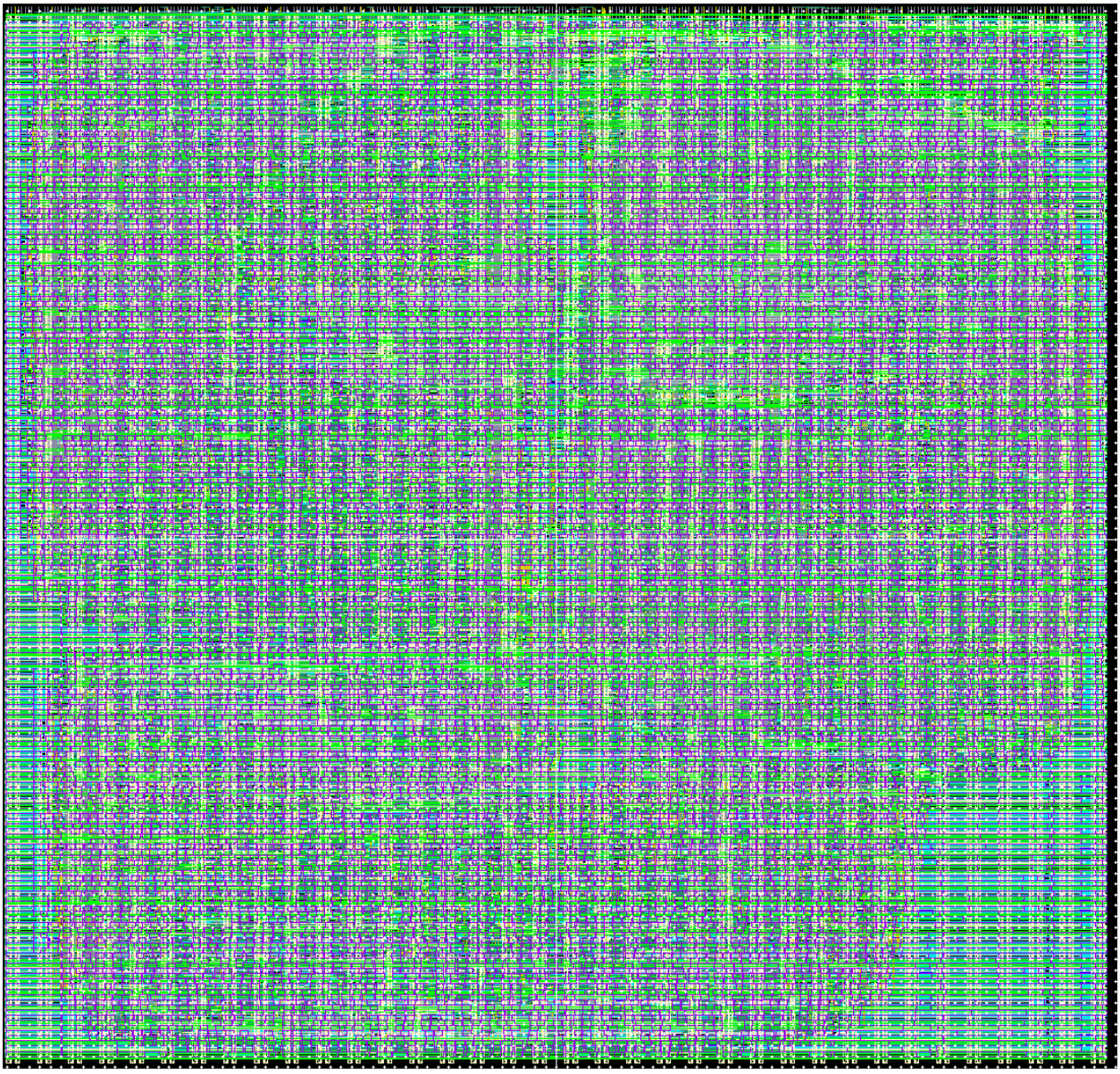

Figure 63: Structural High-performance Multiplier Layout

The *high-performance structural* Montgomery modular multiplier follows the same 4-stage pipelining structure illustrated in Figure 59. Manually repartitioning the logic more evenly across the stages and achieving the free speedup associated with register retiming would require significant design time. Without the assistance of EDA tools, designers must use the natural

separations in the logic for pipelining, as was done in Figure 59. The same HDL, utilizing array multipliers and RCAs, was used as the starting point. Without timing constraints or the ability to target the critical paths in the circuit, all MTNCL cells in the *structural high-performance* multiplier were maximized prior to buffering to maximize throughput.

The *synthesized* design was generated from the same RTL specification used for the 1-stage *low-power* variant. Instead of explicitly partitioning the logic using the available three pipeline registers, all three were behaviorally instantiated at the end of the combinational logic. This simplifies the RTL, requiring less design time, while granting the ability to specify an arbitrary number of pipeline registers. With register retiming enabled during single-rail synthesis, the user can customize the pipeline granularity to meet their performance needs while minimizing unnecessary registers and power consumption. Although the boundary behavior of asynchronous circuits is constant, which provides full pipeline flexibility, the number of pipeline stages was set at four to enable direct comparison with the *structural* design. Genus was given a *500 ps* clock period during single-rail synthesis, resulting in *329 ps* of negative slack, indicating that Genus was unable to optimize the circuit further with the current configuration. The precise architectures Genus selected for the arithmetic components is unclear, though the timing paths suggest the presence of Wallace tree adders.

Beginning from the same netlist, the *synthesized* design was buffered using the same *110 ps* max transition constraint, while the timing constraints guided the buffering of the *optimized* implementation. The *I paths* were set to *2,190 ps*, resulting in a small amount of positive slack on the worst path.

5.2.3.2 Structural vs. Synthesized

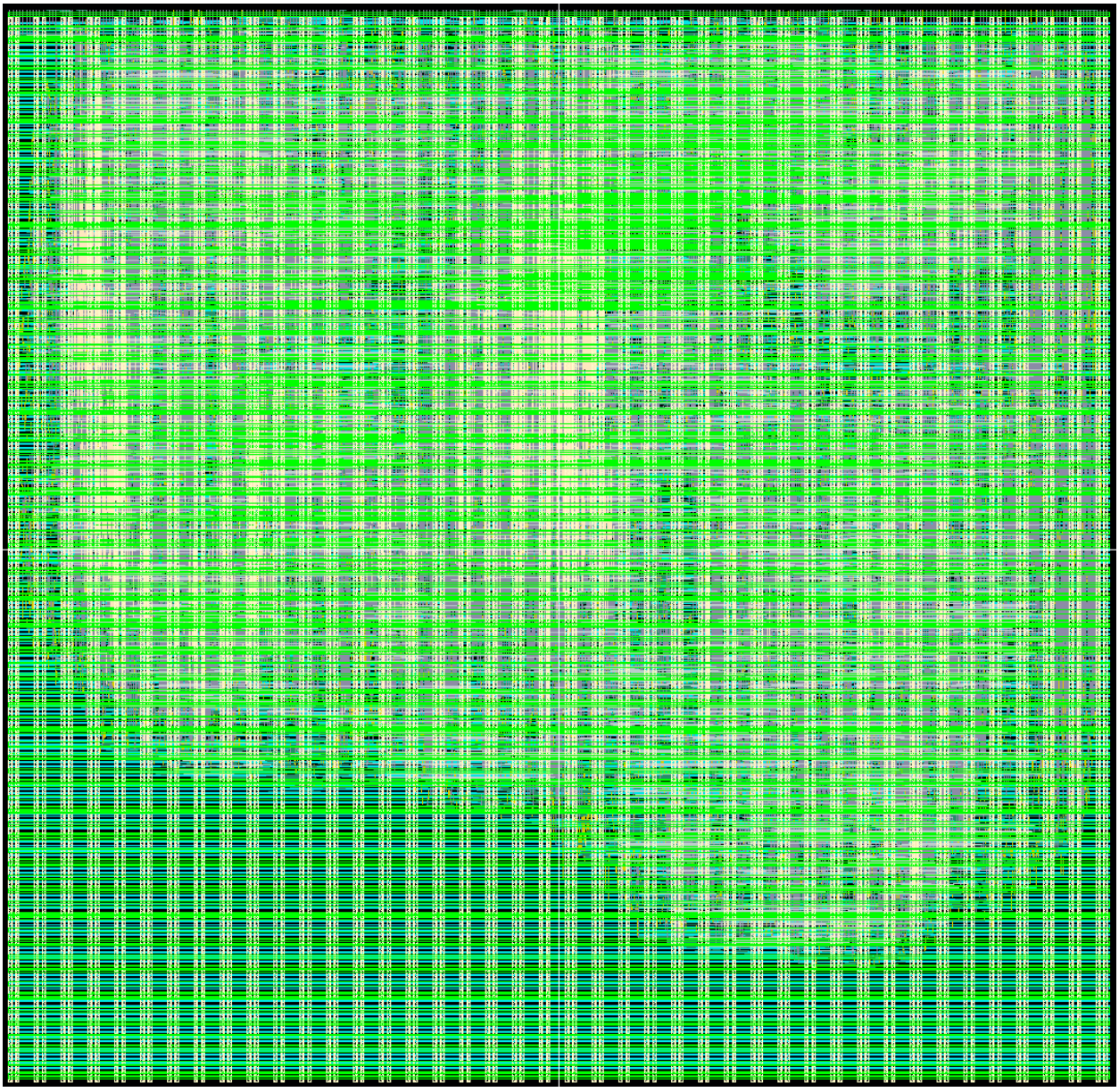

Figure 64: Synthesized High-performance Multiplier Layout

Table 24: Structural vs. Synthesized High-performance Multiplier Metrics

	Structural	Synthesized	Percent Improvement
Average Frequency (MHz)	127.7	178.1	39.5%
Average Active Energy per Operation (pJ)	260.50	246.51	5.4%
EDP (pJ*ns)	2039.72	1383.77	32.2%
Leakage Power (µW)	222.400	204.133	8.2%
Area (µm²)	171298	189438	-10.6%

Table 24 delivers a comparison of the collected metrics for the *structural* and *synthesized* 4-stage *high-performance* Mongomery modular multipliers. Like the *low-power* variations, the synthesis flow significantly boosted performance—in this case by 39.5%. To elucidate where the throughput gain comes from, Table 25 includes the *I path* values for both designs across all four stages. The first observation comes from a comparison of the summation of each design's path delays. Through its architectural selection and intense logical optimization, Genus improved this sum by 48%. The second observation relates to the impact of register retiming. Whereas the worst stage delay of the structural design is *9,172 ps*, Genus halved this value to *4,503 ps*. While the variation of stage delay in the *structural* design is *5,478 ps*, the *synthesized* design has far less variation at *1,210 ps*, a 78% reduction. Most of the stage delays improved by about 50%, with the slight degradation of the stage four delay being inconsequential since it was already significantly faster than the others. As described in Section 5.2.1, the first three stages of the *structural* design utilize the exact same 32×32 array multiplier. Stage two operates faster than its neighbors because the upper half of the product is truncated in the algorithm, and MTNCL's completion detection capitalizes on this shorter critical path.

Table 25: Structural vs. Synthesized High-performance Multiplier Stage Delays

	Structural (ps)	Synthesized (ps)	Improvement
Stage 1	9172	4503	51%
Stage 2	6901	3615	48%
Stage 3	9445	3293	65%
Stage 4	3694	3723	-1%
Total	29212	15134	48%
Range	5751	1210	79%

Despite significant performance improvements, the *synthesized* design consumes 5.4% less active energy per operation and 8.2% less power while idle. Together, the speedup and active energy savings create an EDP reduction of 32.2%. To help explain these savings, especially in light of the 10.6% increase in design area, Table 26 and Table 27 show the high-level composition of both circuits along with each category's switching activity. While the tables *do* indicate that the total switching activity of the *synthesized* design is lower than the *structural* alternative, the reduction is 29% which is far higher than the measured 5.4% active energy savings. As explained previously, these tables hold approximate switching activity, omitting certain details such as parasitic routing capacitance. Although the *synthesized* multiplier only has 10.6% more area than the *structural* version, it has 67% more cells. This jump increased the amount of routing within the design, and a review of the Innovus logs shows that the *synthesized* design has 55% more total wire length. The increase in associated routing capacitance would counterbalance the reduction in cell-level switching presented in the two tables.

The leakage power improvement also warrants elaboration. Despite the area increase, the post-layout transistor-level simulation indicates that the *synthesized* multiplier consumes 8.2%

less static power while idle. This is partly due to the *synthesized* design's use of a large number of small cells, despite a 4× increase in sleep drivers.

Table 26: Structural High-performance Multiplier Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	0	0	0.5	0.5	0
MTNCL X0P7	0	0	0.7	0.5	0
MTNCL X1	0	0	1	0.5	0
MTNCL X2	19786	159506	2	0.5	159506
BUF	1118	11757	1	1	11757
INV	0	0	1	1	0
Total	20904	171263	0	0	171263

Table 27: Synthesized High-performance Multiplier Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	6399	33552	0.5	0.5	8388
MTNCL X0P7	7175	40477	0.7	0.5	14167
MTNCL X1	5986	32796	1	0.5	16398
MTNCL X2	9537	69898	2	0.5	69898
BUF	5325	11756	1	1	11756
INV	476	920	1	1	920
Total	34898	189398	0	0	121527

A review of the total *sleep* driver area reveals a minor increase of 7.8%. Further, all the MTNCL cells in the *structural* design are the maximum size and highest transistor density, *X2*. In contrast, Genus sized the MTNCL cells carefully, using each size in approximately equal proportions across the design. The combination of a 10.6% increase in area and a 67% higher cell count indicates that Innovus, and primarily Genus, are utilizing the design area very efficiently. Due to the significant throughput increase alone but further emphasized by the

improvement to both types of energy efficiency, the *synthesized* design is the clear choice between the two for *high-performan*ce applications.

5.2.3.3 Synthesized vs. Optimized

Figure 65: Optimized High-performance Multiplier Layout

Table 28: Synthesized vs. Optimized High-performance Multiplier Metrics

	Synthesized	Optimized	Percent Improvement
Average Frequency (MHz)	178.1	252.0	41.4%
Average Active Energy per Operation (pJ)	246.51	279.80	-13.5%
EDP (pJ*ns)	1383.77	1110.40	19.8%
Leakage Power (µW)	204.133	276.105	-35.3%
Area (µm²)	189438	197688	-4.4%

Table 28 showcases a comparison of the metrics for the *synthesized* and *optimized high-performance* Mongomery modular multipliers. To facilitate further discussion about the 41.4% improvement in performance, the worst-case *I path* delay for each stage of both circuits has been tabulated in Table 29. The throughput boost can be attributed to an average 43% reduction in *I path* delay across all stages. Aside from improved performance, one of the most significant trends demonstrated by Table 28 is the 94% lower variation of the *I path* delays in the *optimized* design. It should be noted that *2,150 ps* is not the maximum speed each stage can achieve if optimized individually. Instead, Innovus has intelligently absorbed the positive slack in the faster stages through power optimizations. Given the performance bottleneck in MTNCL circuits elucidated by Equation 2, this is precisely what the MTNCL circuit designer should be seeking.

Comparing the same-stage *I paths* of both designs offers additional insights. Notably, it confirms that relying solely upon max transition constraints is not an advisable method of achieving high performance. In response to this constraint, Innovus was overzealous with its buffering, resulting in an added delay of *2,604 ps* from inverters and buffers in the first stage *I path* of the *synthesized* design.

Table 29: Synthesized vs. Optimized High-performance Multiplier Stage Delays

	Synthesized (ps)	Optimized (ps)	Improvement
Stage 1	4503	2150	52%
Stage 2	3615	2184	40%
Stage 3	3293	2118	36%
Stage 4	3723	2110	43%
Total	15134	8562	43%
Range	1210	74	94%

In contrast, the *I path* of the same stage in the *optimized* design clearly indicates that the fastest path can be achieved with *417 ps* of driver delay. This large discrepancy does not necessarily imply that a lower max transition constraint value would result in a noticeably faster *synthesized* multiplier. Instead, the path delay may shift from a large number of buffers to fewer buffers alongside slower MTNCL gates. Additionally, it appears that Genus was unable to accurately predict the final critical path in the circuit, as demonstrated by the far higher occurrence of *X0P5* and *X0P7* gates in the *I paths* for the *synthesized* design. Since completion detection, registration, and *sleep* trees implementation occur after register retiming in the single-rail synthesis step, some divergence between synthesis and physical implementation should be expected.

To shed light on energy-related metrics, Table 27 and its counterpart for the *optimized* variation in Table 30 are included below. By leveraging timing constraints to identify the critical paths in the design, Innovus could effectively reduce the average MTNCL gate size. However, Innovus doubled the total area devoted to driver cells to satisfy the reliability constraints and fine tune the *sleep* trees. Unlike the *structural* and *synthesized high-performance* multipliers discussed previously, the *optimized* variation consumes 13.5% more active energy per

operation—a trend opposite to what the switching activity tables would suggest. Along with the previously discussed approximations intrinsic to the tables, an additional factor is the assumption that all the buffers and inverters are used to construct *sleep* trees and thus have twice the switching activity as the average data signals. However, a large proportion of the driver cells in the *synthesized* multiplier are positioned between the MTNCL gates and thus have lower switching activity than the table assumes. Consequently, the total switching calculation in Table 27 is overly pessimistic.

Table 27: Synthesized High-performance Multiplier Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	6399	33552	0.5	0.5	8388
MTNCL X0P7	7175	40477	0.7	0.5	14167
MTNCL X1	5986	32796	1	0.5	16398
MTNCL X2	9537	69898	2	0.5	69898
BUF	5325	11756	1	1	11756
INV	476	920	1	1	920
Total	34898	189398	0	0	121527

Table 30: Optimized High-performance Multiplier Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	9083	49654	0.5	0.5	12413
MTNCL X0P7	6122	33911	0.7	0.5	11869
MTNCL X1	7532	40646	1	0.5	20323
MTNCL X2	6360	47753	2	0.5	47753
BUF	7511	20739	1	1	20739
INV	1533	4944	1	1	4944
Total	38141	197646	0	0	118041

Although active energy degraded by 13.5%, the throughput boost of 41.4% led to a 19.8% reduction in EDP, indicating that neither constituent metric is sacrificed to improve the other. The doubling of high-leakage driver area in the *optimized* design led to a 35.3% increase in static power dissipation. Despite the higher active and static power consumption, the *optimized* design is the clear winner for *high-performance* applications due to the 41.4% increase in throughput and addition of reliability.

5.2.3.4 Higher Pipeline Granularity

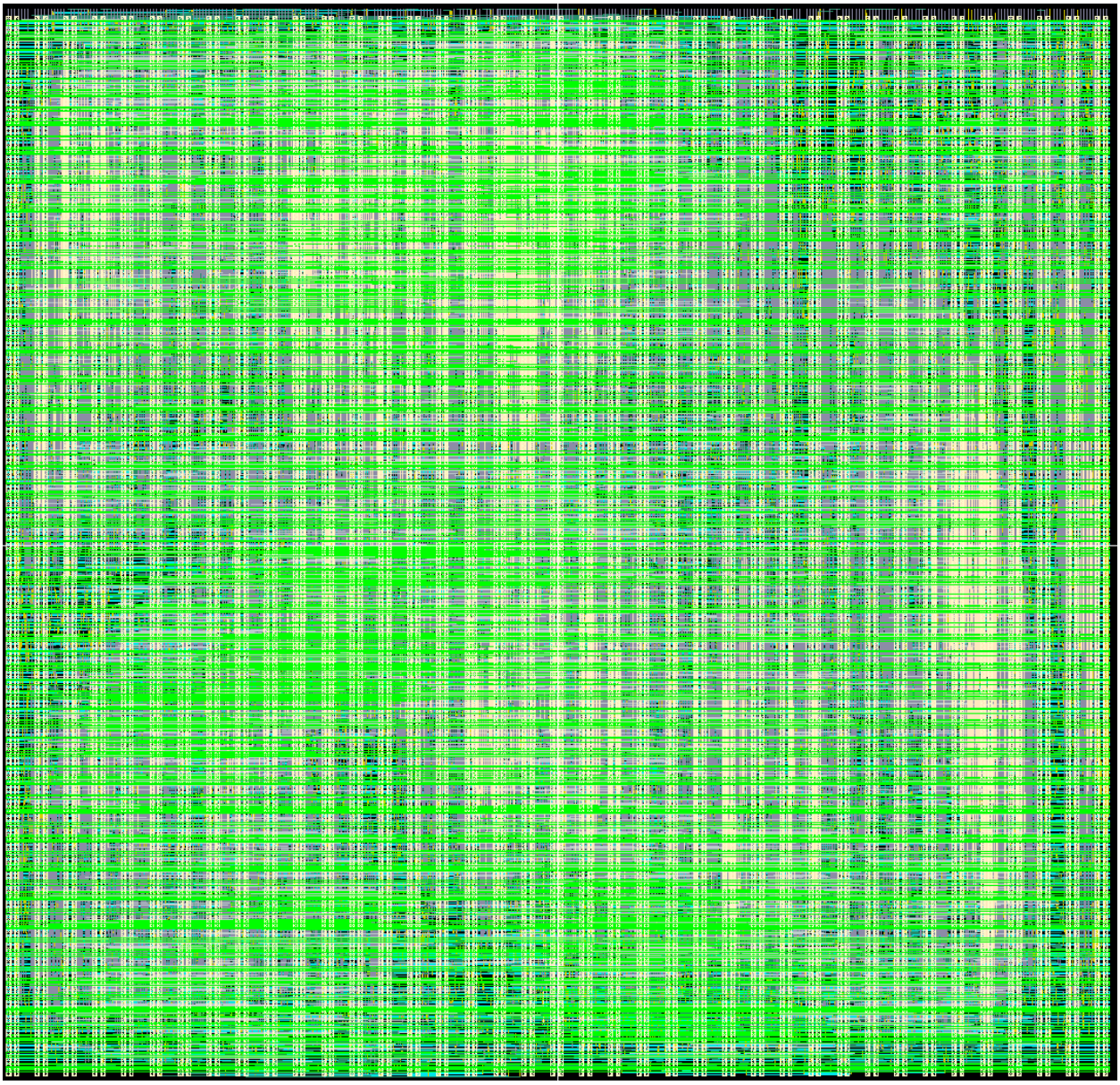

Figure 66: Synthesized 8-stage High-performance Multiplier Layout

This section has been included to explore the effects of higher pipeline granularity in the *high-performance* Montgomery modular multiplier, an optimization exclusive to synthesized circuits using register retiming. Adjusting pipeline granularity in the developed synthesis flow is trivial, while doing the same in structural design is quite the opposite. As laid out in Section 5.2.3.1, the RTL for the Montgomery modular multiplier was written to be generic with respect

to the number of pipeline registers. Given this design flexibility, synthesizing an 8-stage version was as simple as changing the parameter and specifying tighter clock frequencies. Genus then repartitioned the logic evenly across the eight stages.

Table 31: 4-stage vs. 8-stage Synthesized High-performance Multiplier Metrics

	4-stage	8-stage	Percent Improvement
Average Frequency (MHz)	178.1	200.3	12.4%
Average Active Energy per Operation (pJ)	246.51	288.83	-17.2%
EDP (pJ*ns)	1383.77	1442.09	-4.2%
Leakage Power (µW)	204.133	196.310	3.8%
Area (µm²)	189438	207028	-9.3%

Table 31 allows for a comparison between the 4-stage and 8-stage *synthesized* implementations. The two metrics requiring elaboration are the throughput and leakage power. From a theoretical and synchronous perspective, assuming zero flip-flop propagation delay and no clock skew, one would expect the throughput to double given the doubling of pipeline stages. However, the post-layout transistor-level simulations indicate a far less significant 12.4% performance boost. As will be expounded upon later in the section, this significant divergence from linearity is partly intrinsic to the MTNCL architecture. Additionally, the *I paths* from both designs underscore the disconnect between synthesis and physical implementation of MTNCL circuits, i.e., Genus cannot accurately predict what the final critical paths will be after placement and routing.

The cell composition of both *synthesized* designs has been listed in Table 27 and Table 32. Degradation of the active energy per operation and area are to be expected given the additional

four registers and completion detection units. Interestingly, no precise explanation is available for the counterintuitive leakage power improvement of 3.8%. Previous sections have already elaborated on the approximations made in the calculations of these tables. Furthermore, the *synthesized* circuits are buffered unconventionally according to max transition constraints and contain an abnormally large number of buffers between the combinational cells. The location of these buffers and inverters also has a noticeable impact on their leakage current, as drivers in the datapath typically pass *0* while drivers in the *sleep* trees tend to pass *1*. Given these considerations, a slight inverse trend in static power for two very similar circuits is plausible. Although the exact cause(s) remain unclear, the post-layout transistor-level simulations provide extremely high accuracy for this and the other relevant measurements.

Table 27: Synthesized 4-stage High-performance Multiplier Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	6399	33552	0.5	0.5	8388
MTNCL X0P7	7175	40477	0.7	0.5	14167
MTNCL X1	5986	32796	1	0.5	16398
MTNCL X2	9537	69898	2	0.5	69898
BUF	5325	11756	1	1	11756
INV	476	920	1	1	920
Total	34898	189398	0	0	121527

Table 32: Synthesized 8-stage High-performance Multiplier Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	4905	24997	0.5	0.5	6249
MTNCL X0P7	6810	37775	0.7	0.5	13221
MTNCL X1	4814	26919	1	0.5	13459
MTNCL X2	14104	101455	2	0.5	101455
BUF	6693	14421	1	1	14421
INV	665	1372	1	1	1372
Total	37991	206938	0	0	150177

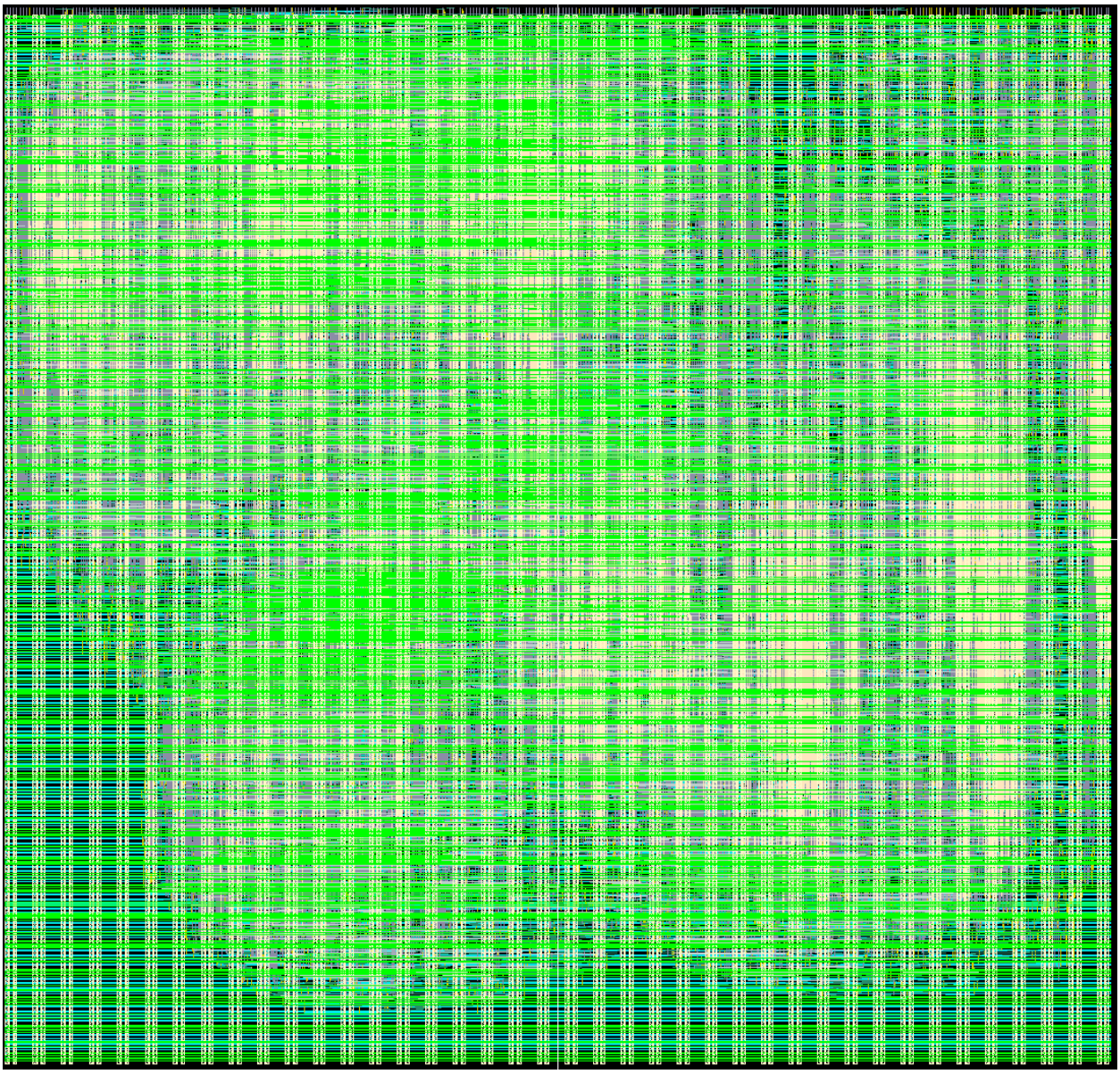

Figure 67: Optimized 8-stage High-performance Multiplier Layout

Table 33: 4-stage vs. 8-stage Optimized High-performance Multiplier Metrics

	4-stage	8-stage	Percent Improvement
Average Frequency (MHz)	252.0	306.3	21.6%
Average Active Energy per Operation (pJ)	279.80	352.10	-25.8%
EDP (pJ*ns)	1110.40	1149.57	-3.5%
Leakage Power (µW)	276.105	385.096	-39.5%
Area (µm²)	197688	226174	-14.4%

Table 33 compares the *optimized* 4-stage and 8-stage *high-performance* Montgomery modular multipliers. The introduction of four additional pipeline stages boosted the fastest 4-stage Montgomery modular multiplier presented in this chapter thus far by an additional 21.6%, leading to multiple intriguing observations. First, timing constraints are a prerequisite to unlocking the full benefits of increased pipelining. This is demonstrated by the 12.4% throughput boost in the *synthesized* designs compared to the 21.6% boost in the *optimized* versions.

As was the case for the *synthesized* alternatives, theory would suggest an approximate doubling of circuit performance with the finer pipeline granularity. Table 34 presents the *I path* delays for both *optimized* circuits, where the approximate 29% speedup to the first four stages accounts for the overall throughput boost. Interestingly, the total *I path* delay through the 8-stage implementation is 41% longer than the 4-stage alternative. Both the modest improvement to individual stage delay and this overall increase can be understood by comparing a full *I path* from both circuits, facilitated by Figure 68 and Figure 69 below. As expected, the number of gates in the path decreased with the transition to eight stages. The paths support one of the central assertions of this work: the *sleep* tree must be implemented *carefully* to ensure it does not bottleneck the circuit. While we would expect the first MTNCL gate in each path to be a register

cell (indicated by an instance name of *block[0-9]*_reg.**), the first cell is actually part of the *combinational logic*. This proves that a slow *sleep* tree will stall the propagation of DATA, as the combinational logic will remain asleep from the previous NULL wave. Nevertheless, the *I* timing constraint provided in Section 3.3.2 addresses these cases, ensuring that Innovus achieves the fastest DATA cycle time attainable.

Table 34: 4-stage vs. 8-stage Optimized High-performance Multiplier Stage Delays

	4-stage	**8-stage**	**Improvement**
Stage 1	2150	1539	28%
Stage 2	2184	1533	30%
Stage 3	2118	1516	28%
Stage 4	2110	1507	29%
Stage 5	N/A	1515	N/A
Stage 6	N/A	1457	N/A
Stage 7	N/A	1543	N/A
Stage 8	N/A	1431	N/A
Total	8562	12041	-41%
Range	74	112	-51%

```
#-------------------------------------------------------------------------------------------
# Timing Point           Arc   Edge  Cell                        Fanout     Delay    Arrival
#                                                                            (ps)       (ps)
#-------------------------------------------------------------------------------------------
  block0_r_comp.th22/z   z     R     (arrival)                        3     3.100      3.100
  comp1_in_finish/z      b->z  F     th22n_with_inv_LVT_X2N           1   134.600    137.700
  FE_PSBC3_OFN315_ko/Y   A->Y  F     BUFH_X9M_A12TL                   8    57.600    195.300
  FE_OCPC3OFN564_ko/Y    A->Y  R     INV_X13B_A12TL                  12    43.400    238.700
  FE_OCPC3_OFN342_ko/Y   A->Y  R     BUFH_X3P5M_A12TL                 1    34.600    273.300
  FE_OCPC2_OFN342_ko/Y   A->Y  F     INV_X5B_A12TL                    6    17.000    290.300
  block0_cck0_wrapped_m  s->z  R     th12m_LVT_HVT_X2N_skewed         2    40.900    331.200
  FE_PSBC3ock0_core_FE_  A->Y  R     BUFH_X3M_A12TL                   5    42.600    373.800
  block0_cRC_1972_0_dup  a->z  R     th22m_LVT_HVT_X1N_skewed         2    80.900    454.700
  block0_cRC_3006_0_dup  c->z  R     th13m_LVT_HVT_X2N_skewed         5    92.100    546.800
  block0_cck0_wrapped_F  a->z  R     th22m_LVT_HVT_X1N_skewed         3   100.200    647.000
  block0_cRC_1685_1/z    b->z  R     thxor0m_LVT_HVT_X2N_skewed       3   111.400    758.400
  block0_cck0_wrapped_F  a->z  R     thxor0m_LVT_HVT_X2N_skewed       4    91.400    849.800
  block0_cck0_wrapped_F  b->z  R     th33m_LVT_HVT_X2N_skewed         3    86.200    936.000
  block0_cRC_1133_0_dup  b->z  R     th24w22m_LVT_HVT_X1N_skewed      2    77.900   1013.900
  block0_cRC_1518_1/z    c->z  R     thxor0m_LVT_HVT_X2N_skewed       4    92.700   1106.600
  block0_cRC_794_0/z     b->z  R     thxor0m_LVT_HVT_X2N_skewed       1    86.500   1193.100
  FE_OCPC2ock0_core_FE_  A->Y  R     BUF_X5M_A12TL                    5    45.700   1238.800
  block0_cck0_wrapped_F  b->z  R     thxor0m_LVT_HVT_X2N_skewed       1    80.500   1319.300
  FE_OCPC1ock0_core_blo  A->Y  R     BUFH_X3M_A12TL                   4    39.500   1358.800
  block0_cck0_wrapped_m  a->z  R     th23w2m_LVT_HVT_X1N_skewed       1    54.100   1412.900
  block0_cRC_505_0/z     a->z  R     th22m_LVT_HVT_X2N_skewed         3    89.900   1502.800
  block0_cRC_757_0_dup/  c->z  R     thxor0m_LVT_HVT_X2N_skewed       1    74.100   1576.900
  block0_c818/z          c->z  R     th24compm_LVT_HVT_X2N_skewed     1    92.900   1669.800
  FE_OFC25k0_core_n_340  A->Y  R     BUF_X7P5M_A12TL                  1    54.800   1724.600
  block0_cRC_1443_0/z    d->z  R     th44m_LVT_HVT_X2N_skewed         1    74.000   1798.600
  FE_OCPC1ock0_core_n_3  A->Y  R     BUFH_X3P5M_A12TL                 1    39.700   1838.300
  block0_cRC_1033_1/z    a->z  R     th33m_LVT_HVT_X2N_skewed         1    72.100   1910.400
  FE_OCPC1ock0_core_FE_  A->Y  R     BUF_X7P5M_A12TL                  1    42.800   1953.200
  block0_c695/z          c->z  R     th44m_LVT_HVT_X2N_skewed         1   101.200   2054.400
  block0_c656/z          a->z  R     th44m_LVT_HVT_X2N_skewed         1    95.800   2150.200
  block0_r_comp.th22/a   a     R     th22n_with_inv_LVT_X2N           1     0.000   2150.200
#-------------------------------------------------------------------------------------------
```

Figure 68: Optimized 4-stage High-performance Multiplier Max DATA Cycle Path

```
#-------------------------------------------------------------------------------------
# Timing Point           Arc   Edge  Cell                       Fanout    Delay    Arrival
#                                                                          (ps)       (ps)
#-------------------------------------------------------------------------------------
  block0__comp.th22/z    z     R     (arrival)                       2    0.300      0.300
  comp1_i_finish/z       b->z  F     th22n_with_inv_LVT_X2N          2  143.200    143.500
  FE_OCPC_OFN419_ko/Y    A->Y  F     BUFH_X16M_A12TL                10   55.300    198.800
  FE_OCPC_OFN1260_n/Y    A->Y  R     INV_X16B_A12TL                  9   38.000    236.800
  FE_OCPC_OFN479_ko/Y    A->Y  F     INV_X13B_A12TL                  6   20.600    257.400
  block0_ck0_wrapped_mo  s->z  R     th22m_LVT_HVT_X1N_skewed        1   44.500    301.900
  block0_RC_156_1/z      b->z  R     th12m_LVT_HVT_X2N_skewed        4   74.900    376.800
  block0_ck0_wrapped_mo  b->z  R     th22m_LVT_HVT_X2N_skewed        3   84.000    460.800
  block0_ck0_wrapped_mo  a->z  R     th22m_LVT_HVT_X0P7N_skewed      1   65.500    526.300
  block0_ck0_wrapped_mo  a->z  R     th12m_LVT_HVT_X1N_skewed        1   49.000    575.300
  block0_RC_742_2/z      a->z  R     th24w22m_LVT_HVT_X2N_skewed     1   66.400    641.700
  block0_RC_1561_1/z     d->z  R     thxor0m_LVT_HVT_X2N_skewed      3   82.800    724.500
  block0_RC_673_1/z      c->z  R     thxor0m_LVT_HVT_X2N_skewed      3   89.500    814.000
  block0_ck0_wrapped_FE  b->z  R     th12m_LVT_HVT_X1N_skewed        1   66.900    880.900
  block0_RC_1576_1/z     a->z  R     th33w2m_LVT_HVT_X2N_skewed      1  114.100    995.000
  block0_RC_670_1/z      b->z  R     th24compm_LVT_HVT_X2N_skewed    1  105.300   1100.300
  block0_3571/z          d->z  R     th44m_LVT_HVT_X2N_skewed        1   71.100   1171.400
  block0_3524/z          d->z  R     th44m_LVT_HVT_X2N_skewed        1   89.400   1260.800
  block0_3481/z          d->z  R     th44m_LVT_HVT_X2N_skewed        1   75.900   1336.700
  block0_RC_1475_1/z     b->z  R     th44m_LVT_HVT_X2N_skewed        1   97.300   1434.000
  block0_RC_246_0/z      d->z  R     th44m_LVT_HVT_X2N_skewed        1  105.500   1539.500
  block0__comp.th22/a    a     R     th22n_with_inv_LVT_X2N          1    0.000   1539.500
#-------------------------------------------------------------------------------------
```

Figure 69: Optimized 8-stage High-performance Multiplier Max DATA Cycle Path

An analysis of these paths also reveals an intrinsic aspect of MTNCL and other asynchronous templates utilizing completion detection. The additional logic included to facilitate handshaking—specifically, the completion detection and *sleep* trees for MTNCL—reduce the throughput gains associated with increasing pipeline granularity. Table 35 supplies a breakdown of each delay, where an additional *100 ps* has been added to the completion detection category to account for the missing *TH22*. Note that the registration does *not* contribute to the path delay. As the number of pipeline stages increases, the proportion of combinational logic diminishes while the support logic increases, demonstrating that pipelining MTNCL circuits has diminishing returns.

Table 35: 4-stage vs. 8-stage High-performance Multiplier DATA Cycle Breakdown

	4-stage		8-stage	
	Delay	**Percentage**	**Delay**	**Percentage**
Registration	0	0%	0	0%
Sleep Distribution	153	7%	114	7%
Combinational Logic	1287	60%	738	48%
Completion Detection	711	33%	688	45%
Totals	2150	100%	1540	100%

5.2.3.5 Summary

Table 36: Structural vs. Optimized High-performance Multiplier Metrics

	Structural	**Optimized**	**Percent Improvement**
Average Frequency (MHz)	127.7	252.0	97.3%
Average Active Energy per Operation (pJ)	260.50	279.80	-7.4%
EDP (pJ*ns)	2039.72	1110.40	45.6%
Leakage Power (µW)	222.400	276.105	-24.1%
Area (µm²)	171298	197688	-15.4%

Table 36 highlights the advantages of using a synthesis flow built upon commercial EDA tools and timing constraints during placement and routing for *high-performance* Montgomery modular multipliers. The data for the 4-stage *optimized* design was selected to enable a direct and fair comparison. Using commercial synthesis tools like Genus provides easy access to a variety of component architectures, intense logical optimization, and register retiming. In addition, the application of timing constraints during physical implementation enables optimization of the critical paths for higher performance and of the non-critical paths for lower power consumption. For 4-stage 32×32 *high-performance* Montgomery modular multipliers, using this flow has the potential to nearly *double* performance. The tradeoffs for this monumental speedup include

relatively minor increases of 7.4% in active energy, 24.1% in leakage power, and 15.4% in design area. Furthermore, the final design can be trusted to function reliably in the PVT corners assessed during signoff timing analysis. Beyond far exceeding the QoR of the equivalent *structural* design, the synthesis flow unlocks at least 2.4× the performance, as demonstrated by the 8-stage designs. For these reasons, *structural* design of this circuit type is not a viable approach in *high-performance* applications.

5.3 AES-256 Cores

5.3.1 Architecture

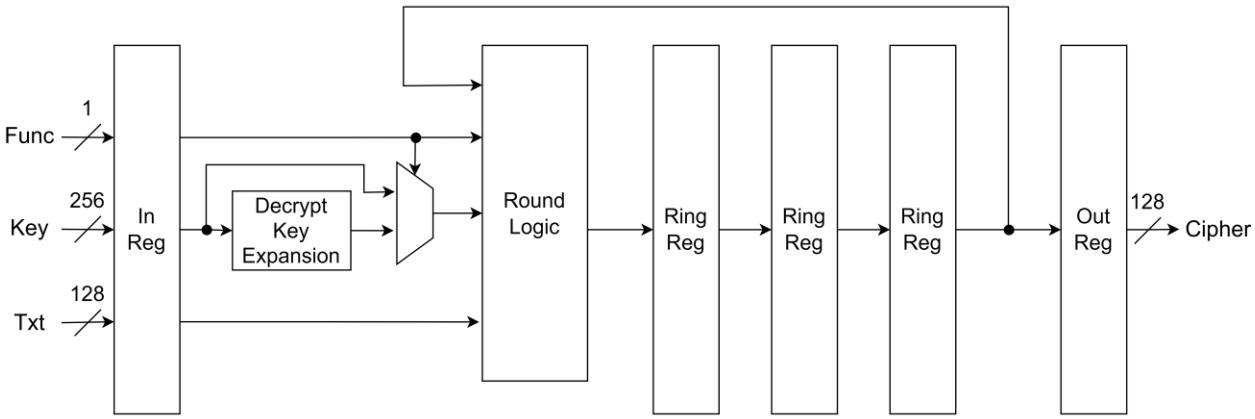

Figure 70: AES-256 Core Architecture

Figure 70 lays out the high-level structure of the electronic codebook (ECB) AES-256 cores implemented as part of this work. The input register captures the 256-bit secret key, 128-bit plaintext or ciphertext, and 1-bit function that specifies whether encryption or decryption is to be performed, respectively. Following key expansion and subsequent function-based key selection, the text, selected key, and function bit enter the round logic. Unlike *high-performance* commercial AES-256 cores, which typically utilize thoroughly pipelined, fully combinational designs, the pre-existing MTNCL *structural* design was intended for a power side-channel attack

project and composed the round logic sequentially. To this end, the figure shows the sequential structure, encompassing the combinational round logic and a three-ring register responsible for holding the current state of the Finite State Machine (FSM). After the standard 14 rounds in the AES-256 algorithm, the new text is latched by the output register before leaving the AES core. Although the internal FSM cycles 14 times per input text, the input and output interfaces only handshake once per DATA wave. To accomplish this, the handshaking for the *Ki* of the ring register was designed to allow the FSM to cycle independently until the end of the last round, at which point the output completion detection drives the handshaking.

The information provided thus far describes both the *structural* MTNCL version and the behavioral RTL used for synthesis; however, these designs are not identical and have some architectural differences. The original behavioral RTL for the AES-256 core was acquired from opencores.org [40] and selected for its availability, simplicity, and sequential implementation. Conversely, the other candidate RTL sets did not support the ECB block cipher mode, were fully combinational, or were excessively complex and difficult to follow. Simplicity was crucial, as it became clear that the selected RTL would require significant modifications to more closely align with the *structural* architecture and permit a viable comparison.

The selected RTL was written for and tested on an Altera Cyclone II FPGA and targeted an Avalon bus interface. As such, the RTL read the text and key from the bus at 32-bits per cycle. This logic was removed to allow the inputs to be provided to the input interface in a single cycle. In pursuit of the highest performance, it appears the RTL designer split the initial key expansion into enough cycles to prevent it from bottlenecking the clock frequency, resulting in each of the subsequent 14 AES rounds taking a single cycle. Specifically, the designer split the key

expansion operation into a 115-cycle FSM. The expanded key is then stored within the AES core until a different key is loaded, ensuring this lengthy expansion operation occurs infrequently.

This key expansion approach presented two challenges for this work. First, creating a 115-cycle MTNCL FSM would severely degrade the overall design throughput. Second, the *structural* AES-256 core performs key expansion for *each* decryption operation, independent of whether the key was changed. Further, the method that the behavioral RTL used to partition the overall circuit operation between the initial key expansion and the rounds differed from the *structural* design. In consideration of these differences, modifications were made to make the two architectures as similar as possible. First, although the key expansion in the *structural* design was implemented with a 7-cycle FSM, the 115-cycle key expansion in the *synthesized* design was unrolled into a single-stage, fully combinational design. Second, the key expansion was performed with each new input set instead of only when the key is changed. Finally, the *structural* design was modified to perform key expansion during both decryption and encryption operations to match the switching activity and critical path of the *synthesized* design.

Since the behavioral RTL targeted an FPGA, the *S-box* was implemented with a lookup table. However, the *structural* MTNCL design employs a combinational *S-box* implementation, as described in [41], to avoid introducing memory macros during physical implementation. During this work, it was discovered that the lookup table implementation yields a fast but very large and power-hungry design. On the other hand, adapting the dual-rail mathematical *S-box* implementation from the *structural* design for usage in the synthesis flow resulted in a slower but far smaller and more power-efficient design. To optimize the design for each application, the lookup table was utilized in synthesis of the *high-performance* AES-256 core, while the *low-power* version adopted the mathematical implementation.

The existing MTNCL synthesis flow only directly supports fully combinational designs, so synthesis of the AES-256 circuit, comprised of both combinational and sequential logic, required more manual intervention. The key expansion and round logic were extracted from the synchronous RTL and synthesized in isolation as combinational blocks. In the absence of a balanced pipeline, this approach allowed for the application of two different clock frequencies and meaningful optimization to both components. A top-level Bash script was written to interleave execution of the various synthesis flow steps for both designs, facilitating logical verification of both circuits as they progressed through the flow. Finally, a single gate-level netlist was constructed through flattening.

5.3.2 Low Power

5.3.2.1 Implementation

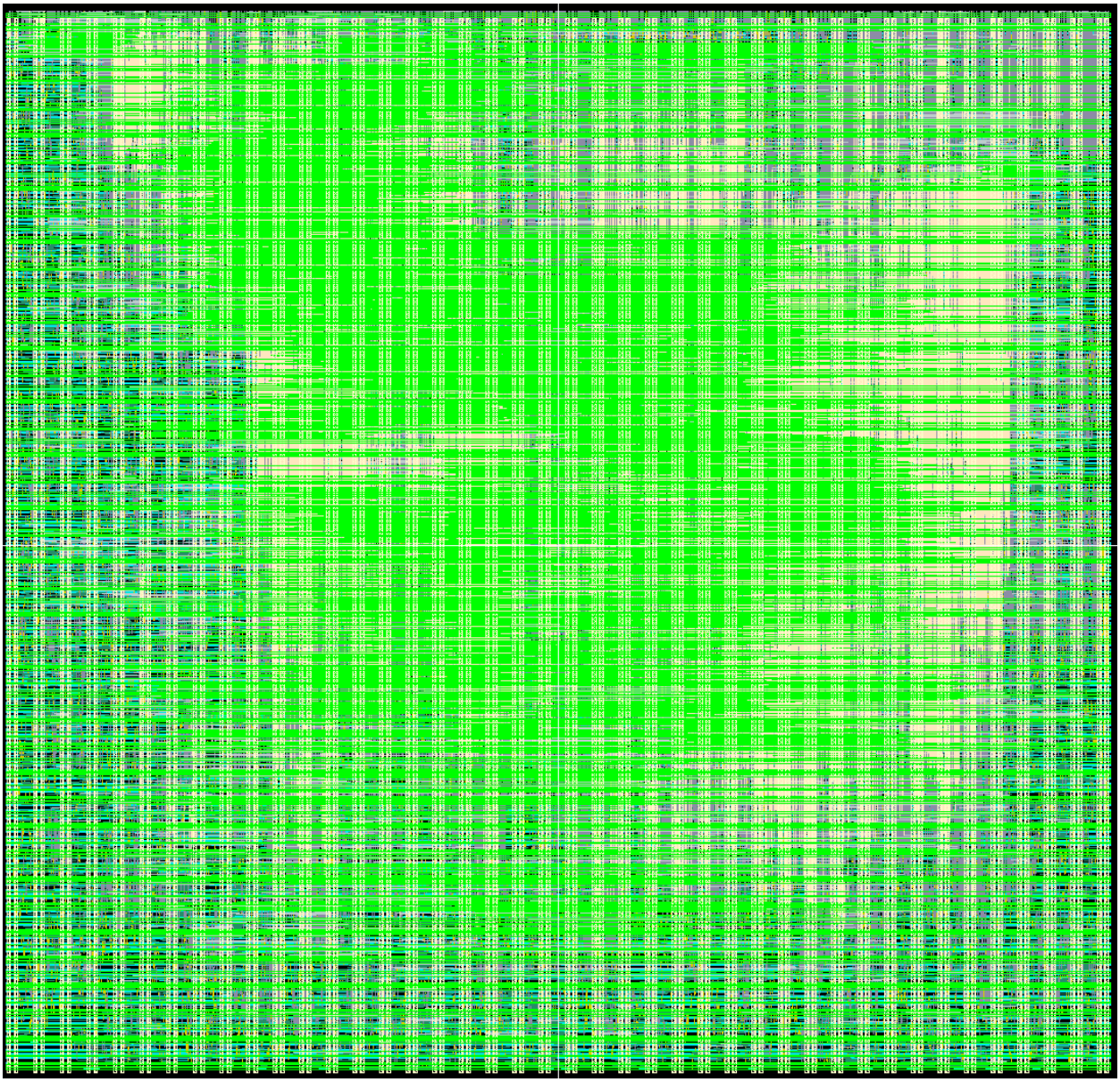

Figure 71: Structural Low-power AES Core Layout

The *structural low-power* AES-256 core follows the same structure depicted in Figure 70. Unlike the adder and multiplier, where a designer can choose from various potential architectures like RCA and Baugh-Wooley, the degree of customization for this design is limited. Apart from the mathematically implemented *S-box* described earlier, the *structural* core directly implements

the AES-256 algorithm, as published by NIST in [38]. The completion detection utilizes a balanced *AND tree*, and all the gates were minimized to reduce power consumption prior to buffering.

The RTL for the behavioral AES-256 core was far more complex than the other designs, consisting of about 35 separate VHDL files, half of which were needed to implement the *S-box* mathematically. Intentionally slow *100 ns* clocks were provided to Genus for single-rail synthesis of the designs, resulting in substantial positive slack and maximal power optimization with little performance concern.

The same strategy was employed when selecting the timing constraints for the *optimized low-pow*er AES-256 core during physical implementation. The *I path* delays through key expansion and round logic were set to *75 ns* and *15.5 ns*, respectively. In return, Innovus uniformly utilized minimum-sized gates.

5.3.2.2 Structural vs. Synthesized

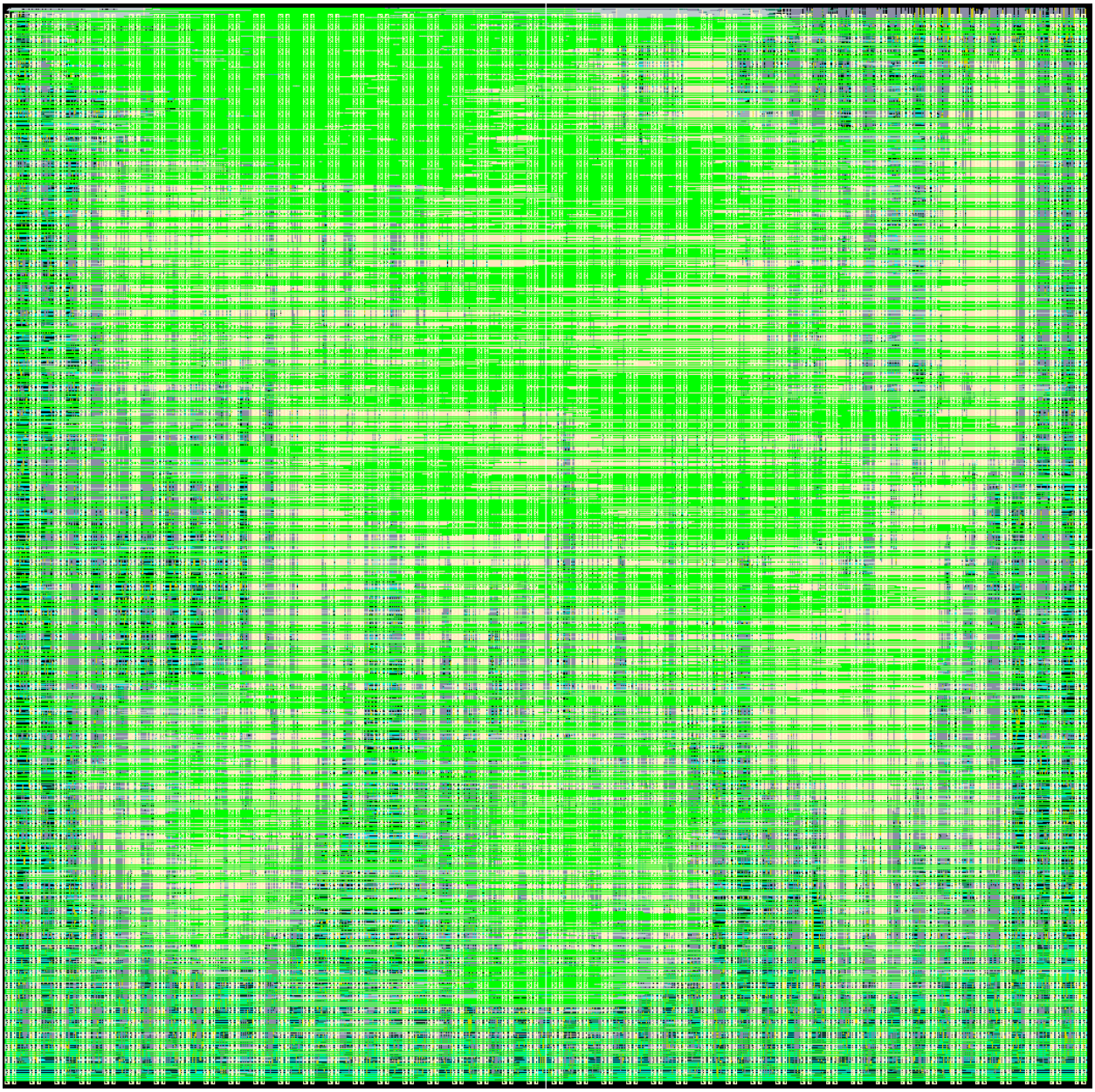

Figure 72: Synthesized Low-power AES Core Layout

Table 37: Structural vs. Synthesized Low-power AES Core Metrics

	Structural	Synthesized	Percent Improvement
Average Frequency (MHz)	4.7	4.0	-15.7%
Average Active Energy per Operation (pJ)	3214.26	2484.22	22.7%
EDP (pJ*ns)	680150	623261	8.4%
Leakage Power (µW)	201.156	152.222	24.3%
Area (µm²)	235842	272208	-15.4%

Table 37 presents the collected metrics for the *structural* and *synthesized low-power* AES-256 cores. The *synthesized* implementation experienced a 16% degradation in overall throughput. While undesirable, the decrease by itself is not concerning given the *low-power* target application of this circuit category. Table 38 supplies rough estimates of each design's total cycle time based on the *I path* delays through the key expansion and round logic, as well as the number of iterations for each component. Counterintuitively, the trend of Table 38 is the inverse of the collected throughput results. The explanation for the inverted trend lies in one of the primary benefits of MTNCL and QDI circuits more broadly: they exhibit *average-case* performance due to the incorporation of multi-rail logic and completion detection. Although the *worst-case* delays in Table 38 indicate that the *synthesized* design should outperform its *structural* counterpart, the *structural* design achieves better *average-case* performance. Unfortunately, Genus approached the synthesis of this circuit as if it was synchronous, reducing delay on the critical paths to improve performance and absorbing positive slack on the non-critical paths to reduce power consumption.

Table 38: Structural vs. Synthesized Low-power AES Core DATA Cycle Breakdown

	Structural	Synthesized
Key Expansion	18.7	75.6
Number of Iterations	7	1
Total Key Expansion	131.1	75.6
Round Logic	20.0	18.5
Number of Iterations	14	14
Total Round Logic	279.3	258.7
Total Cycle Delay	410.5	334.3

Although Table 39 and Table 40 break down the composition of both AES-256 cores, they do not illuminate how Genus and Innovus managed to reduce active energy per operation and static power dissipation by 23% and 24%, respectively, despite the 15% increase in total design area. The *synthesized* core has 27% more cell instances than the *structural* alternative, indicating that Genus used a higher quantity of smaller cells. Like previously presented designs, the tables indicate that the *synthesized* circuit should consume more active energy than its *structural* counterpart. However, by instructing Genus to optimize the circuit for low power consumption, the synthesis program carefully selected energy-efficient cells from the libraries.

Considering both the improvement in active energy and degradation in performance, the *synthesized* design achieved an 8% lower EDP than the *structural* circuit, indicating that the active energy savings are not a simple trade-off for reduced performance. Furthermore, the increases in instance count and total design area in the *synthesized* design can be attributed to the fully unrolled key expansion logic. In summary, the *synthesized* implementation is the superior choice for *low-power* applications given its near 25% improvement in energy efficiency during both active and idle modes.

Table 39: Structural Low-power AES Core Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	34824	218586	0.5	0.5	54647
MTNCL X0P7	0	0	0.7	0.5	0
MTNCL X1	1298	7165	1	0.5	3582
MTNCL X2	0	0	2	0.5	0
BUF	1692	9874	1	1	9874
INV	37	53	1	1	53
Total	37851	235678	0	0	68156

Table 40: Synthesized Low-power AES Core Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	43276	257452	0.5	0.5	64363
MTNCL X0P7	761	4172	0.7	0.5	1460
MTNCL X1	886	4983	1	0.5	2492
MTNCL X2	26	174	2	0.5	174
BUF	1313	2545	1	1	2545
INV	1917	2797	1	1	2797
Total	48179	272124	0	0	73831

5.3.2.3 Synthesized vs. Optimized

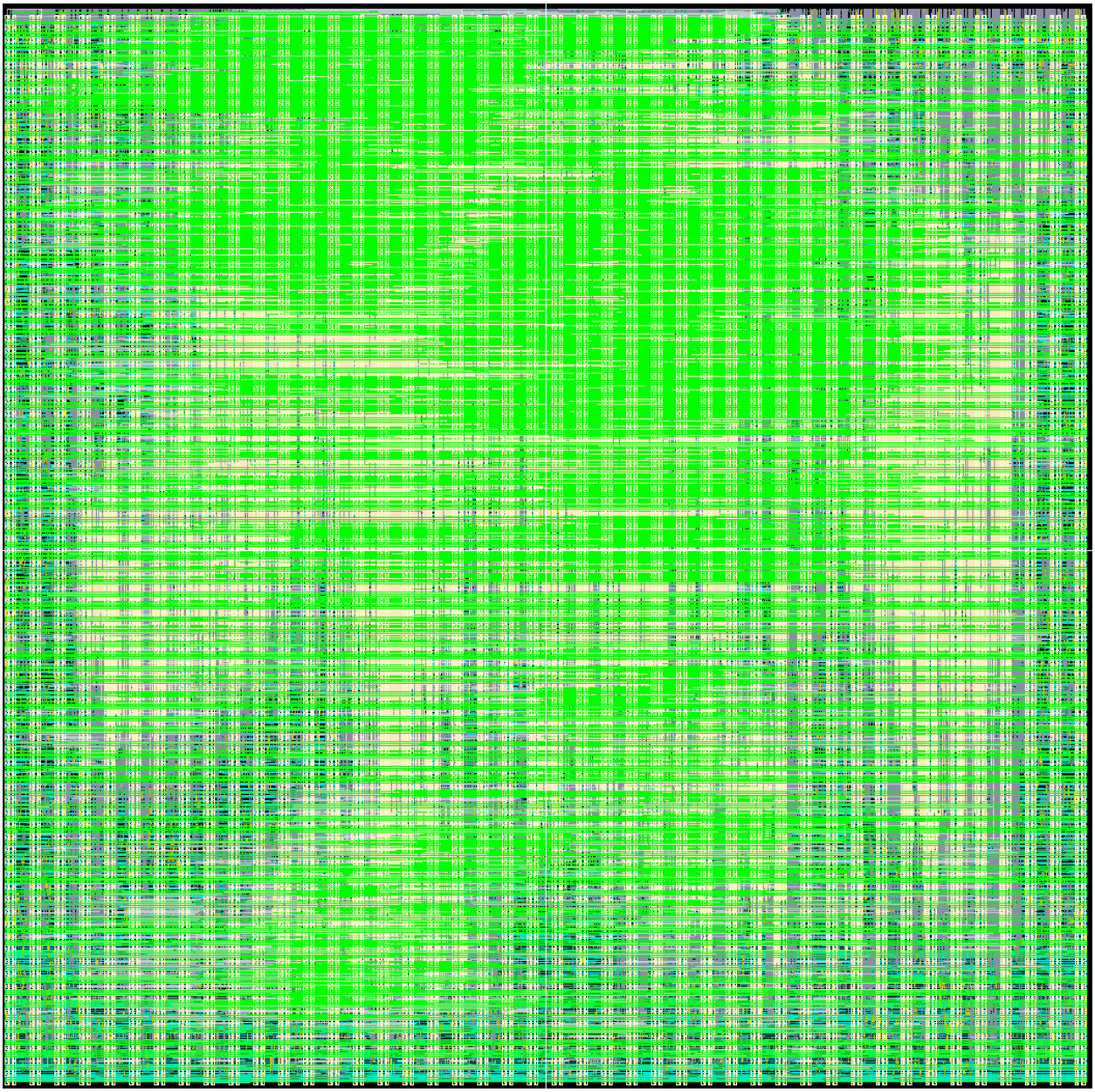

Figure 73: Optimized Low-power AES Core Layout

Table 41: Synthesized vs. Optimized Low-power AES Core Metrics

	Synthesized	Optimized	Percent Improvement
Average Frequency (MHz)	4.0	4.7	17.6%
Average Active Energy per Operation (pJ)	2484.22	2477.21	0.3%
EDP (pJ*ns)	623261	528311	15.2%
Leakage Power (µW)	152.222	160.399	-5.4%
Area (µm²)	272208	272480	-0.1%

Table 41 presents the collected metrics for the *synthesized* and *optimized low-power* AES-256 cores. Although the timing constraints provided to Innovus for placement and routing of the *optimized* cryptographic core were loose—resulting in nanoseconds of positive slack on average—informing the tool of the circuit's critical paths led to a 17.6% increase in throughput, largely due to more effective buffering strategies. Table 42 supplies an approximation of the cycle times based on the *I path* delay through the key expansion and round logic components. The throughput gains primarily stem from a 16% speedup to all 14 rounds of the AES-256 algorithm. The almost *3 ns* penalty of the *synthesized* design is mainly driven by its subpar buffering, with 43% more total buffer delay than the *optimized* design. Moreover, the *optimized* design benefits from Innovus's native optimizations, such as placing cells along the critical paths closer together and optimizing the routing for lower parasitic delay.

Table 42: Synthesized vs. Optimized Low-power AES Core DATA Cycle Breakdown

	Synthesized	Optimized
Key Expansion	75.6	73.1
Number of Iterations	1	1
Total Key Expansion	75.6	73.1
Round Logic	18.5	15.5
Number of Iterations	14	14
Total Round Logic	258.7	217.3
Total Cycle Delay	334.3	290.4

While the overall design area and active energy per operation correspond well, the leakage power increase of 5.4% merits some exploration. Table 40 and Table 43 have been inserted below to enable a side-by-side comparison of design composition, revealing that the *optimized* design has more *X1* and *X2* MTNCL cells than the *structural* alternative. Further, the timing constraints led to a consolidation of the drivers, with a 13.3% reduction in instance count and a 2% increase in area. These two differences arise from the way the timing constraints are calibrated. It was observed that using relatively fast values for *E* and *G* resulted in a noticeable decrease in total driver area. Consequently, when compared with the *synthesized* design, the faster *sleep* distribution and register transition triggered upsizing of the MTNCL cells and *sleep* drivers. Although the active energy did not change meaningfully, the performance increase slashed the EDP by 15.2%. Despite the degraded energy efficiency in response to the application of timing constraints, the reliability imparted by their inclusion outweighs the modest QoR tradeoff.

Table 40: Synthesized Low-power AES Core Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	43276	257452	0.5	0.5	64363
MTNCL X0P7	761	4172	0.7	0.5	1460
MTNCL X1	886	4983	1	0.5	2492
MTNCL X2	26	174	2	0.5	174
BUF	1313	2545	1	1	2545
INV	1917	2797	1	1	2797
Total	48179	272124	0	0	73831

Table 43: Optimized Low-power AES Core Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	43025	256407	0.5	0.5	64102
MTNCL X0P7	556	2712	0.7	0.5	949
MTNCL X1	1285	7279	1	0.5	3640
MTNCL X2	83	546	2	0.5	546
BUF	1765	3606	1	1	3606
INV	1034	1846	1	1	1846
Total	47748	272395	0	0	74688

5.3.2.4 Summary

Table 44: Structural vs. Optimized Low-power AES Core Metrics

	Structural	Optimized	Percent Improvement
Average Frequency (MHz)	4.7	4.7	-0.8%
Average Active Energy per Operation (pJ)	3214.26	2477.21	22.9%
EDP (pJ*ns)	680150	528311	22.3%
Leakage Power (µW)	201.156	160.399	20.3%
Area (µm²)	235842	272480	-15.5%

Table 44 illustrates the QoR impacts resulting from the application of a robust synthesis flow and timing constraints during physical implementation for a *low-power* AES-256 core. Given the target application, the 1% reduction in performance and 15.5% inflation in design area are not problematic. Instead, the 22% reduction to both active energy and leakage power demonstrates that the *optimized* design is better suited to the target application. Genus effectively simplified the circuit logic and carefully selected cells from the library to minimize power consumption, while Innovus intelligently buffered the design, maintained performance, and provided reliable operation to the *optimized* circuit.

5.3.3 High Performance

5.3.3.1 Implementation

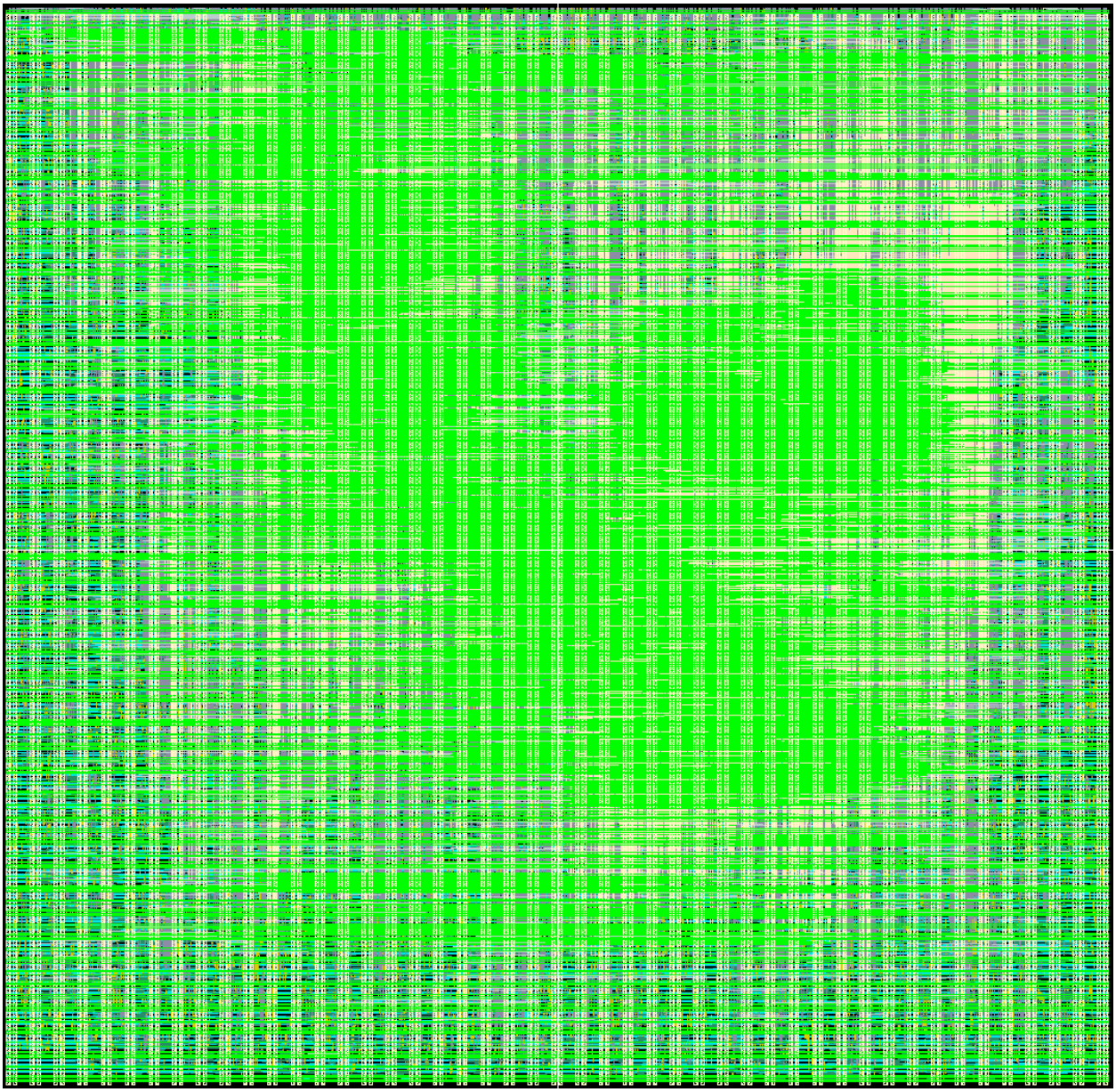

Figure 74: Structural High-performance AES Core Layout

Demonstrating the practical limitations involved in *structural* circuit design, the sole difference between the *structural* AES-256 cores was gate sizing. Whereas the *low-power* design exclusively utilizes minimized MTNCL gates, all the gates were upsized to the maximum (*X2*) in

the *high-performance* implementation to enhance throughput. Without the ability to identify the circuit critical paths, this coarse-grained, indiscriminate sizing approach is the only option available when structurally designing circuits.

To reiterate, unlike the adders and multipliers evaluated and presented in this work, the *high-performance* and *low-power* AES-256 cores were not synthesized from the exact same RTL. The *low-power* RTL employed a mathematical *S-box*, while the *high-performance* RTL implements the *S-box* using a lookup table. Specifically, the lookup table was given to Genus to implement and optimize *combinationally* instead of placing the *S-box* in a memory macro during physical implementation. After elaboration, Genus treated the *S-box* as a complex set of Boolean equations and spent considerable time intensely optimizing the instances for low delay. Single-rail synthesis of the key expansion and round logic were carried out with *4 ns* and *1 ns* clock periods, respectively. This resulted in hundreds of picoseconds of negative slack, indicating that Genus was unable to optimize the circuits further. Table 45, shown below, provides the post-physical-implementation logic depth through the key expansion and round logic of the *low-power* and *high-performance* AES-256 cores to highlight the effects of the distinct *S-box* RTL specifications and Genus's optimization strategies.

Table 45: Low-power vs. High-performance AES Core Logic Depth

	Low-power	**High-performance**	**Improvement**
Key Expansion	435	121	72%
Round Logic	59	24	59%

The performance-related constraints for the *optimized* design were also set to very low values, ensuring that Innovus optimized the design for the highest performance. The *I* constraints for the key expansion and round logic were set to *13.6 ns* and *4.5 ns*, respectively.

5.3.3.2 Structural vs. Synthesized

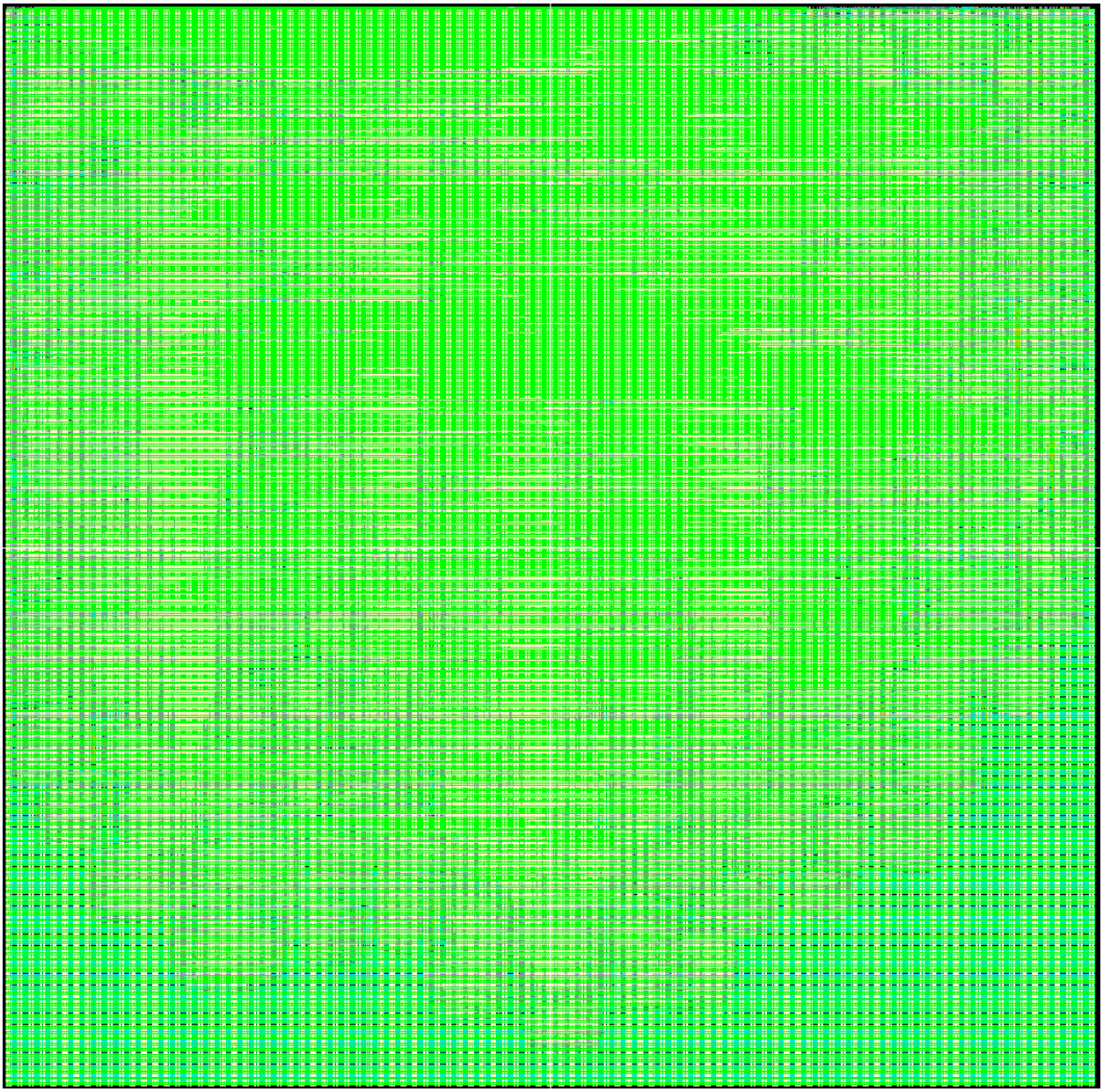

Figure 75: Synthesized High-performance AES Core Layout

Table 46: Structural vs. Synthesized High-performance AES Core Metrics

	Structural	Synthesized	Percent Improvement
Average Frequency (MHz)	6.6	6.7	2.5%
Average Active Energy per Operation (pJ)	5898.54	8995.57	-52.5%
EDP (pJ*ns)	897606	1335578	-48.8%
Leakage Power (µW)	410.573	1332.963	-224.7%
Area (µm²)	301031	1140041	-278.7%

Table 46 furnishes the design metrics for the *structural* and *synthesized high-performance* AES-256 cores. Despite the lookup table *S-box* and Genus's intense logical optimizations, throughput only increased by a marginal 2.5%. Table 47 has been supplied below with an approximate breakdown of each designs' total cycle time as a function of the *I path* delays through the key expansion and round logic, where the projected cycle time indicates that the measured performance should have increased by 44%. The meager 2.5% improvement is an excellent demonstration of MTNCL's *average-case* performance. Genus, guided by synchronous design practices and tight timing constraints, focused heavily on the *worst-case* delay. This utility views positive slack on non-critical paths as free power optimization, worsening the *average-case* delay in the final MTNCL circuit post synthesis.

Table 47: Structural vs. Synthesized High-performance AES Core DATA Cycle Breakdown

	Structural	Synthesized
Key Expansion	11.9	23.7
Number of Iterations	7	1
Total Key Expansion	83.4	23.7
Round Logic	11.8	10.6
Number of Iterations	14	14
Total Round Logic	164.5	148.4
Total Cycle Delay	247.9	172.0

Table 48 and Table 49 have been included below to provide greater visibility into the substantial increases in area, leakage power, and average active energy per operation. The lookup table *S-box* implementation and associated intense logical optimizations led to a 342% increase in MTNCL cell count. Comparing this factor with the relatively small 52.5% increase in active energy per operation demonstrates how efficient Genus is when it selects and sizes the cells during synthesis. *Structural* circuit designers, constrained by time, typically select the MTNCL threshold gate that most directly matches the desired functionality when composing the circuit, with no consideration for the PPA. Furthermore, Innovus likely spaced routes apart to minimize coupling capacitance and prudently managed fanout and capacitance when splitting large nets during buffering. Unlike the custom buffering script, Innovus also accounts for parasitics while buffering according to the max transition constraint.

Table 48: Structural High-performance AES Core Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	0	0	0.5	0.5	0
MTNCL X0P7	0	0	0.7	0.5	0
MTNCL X1	1298	7165	1	0.5	3582
MTNCL X2	34824	270555	2	0.5	270555
BUF	2638	23094	1	1	23094
INV	37	53	1	1	53
Total	38797	300867	0	0	297284

Table 49: Synthesized High-performance AES Core Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	8898	52918	0.5	0.5	13229
MTNCL X0P7	39070	205811	0.7	0.5	72034
MTNCL X1	56930	290936	1	0.5	145468
MTNCL X2	66775	461988	2	0.5	461988
BUF	51691	110229	1	1	110229
INV	10203	18075	1	1	18075
Total	233567	1139956	0	0	821023

Although Innovus has significant control over the design's final active energy usage, its choices for optimizing leakage power are more limited. Excluding more advanced features like block-level power gating and voltage islands, the only optimization available to Innovus is downsizing gates and modifying the design's buffering. For this reason, the 224% increase in static power consumption is an unsurprising consequence of the 342% rise in MTNCL gate count. Further, while the *structural* design devotes 8% of its area to drivers, the *synthesized* design dedicates 11%, resulting in a 5.54× increase in driver area due to the 278.7% growth in overall design area. The substantial increase in these high leakage cells explains the spike in

static power. While the *synthesized* design is marginally faster than the *structural* alternative, the severe penalties in the other design metrics make it prohibitively expensive, even in *high-performance* applications.

5.3.3.3 Synthesized vs. Optimized

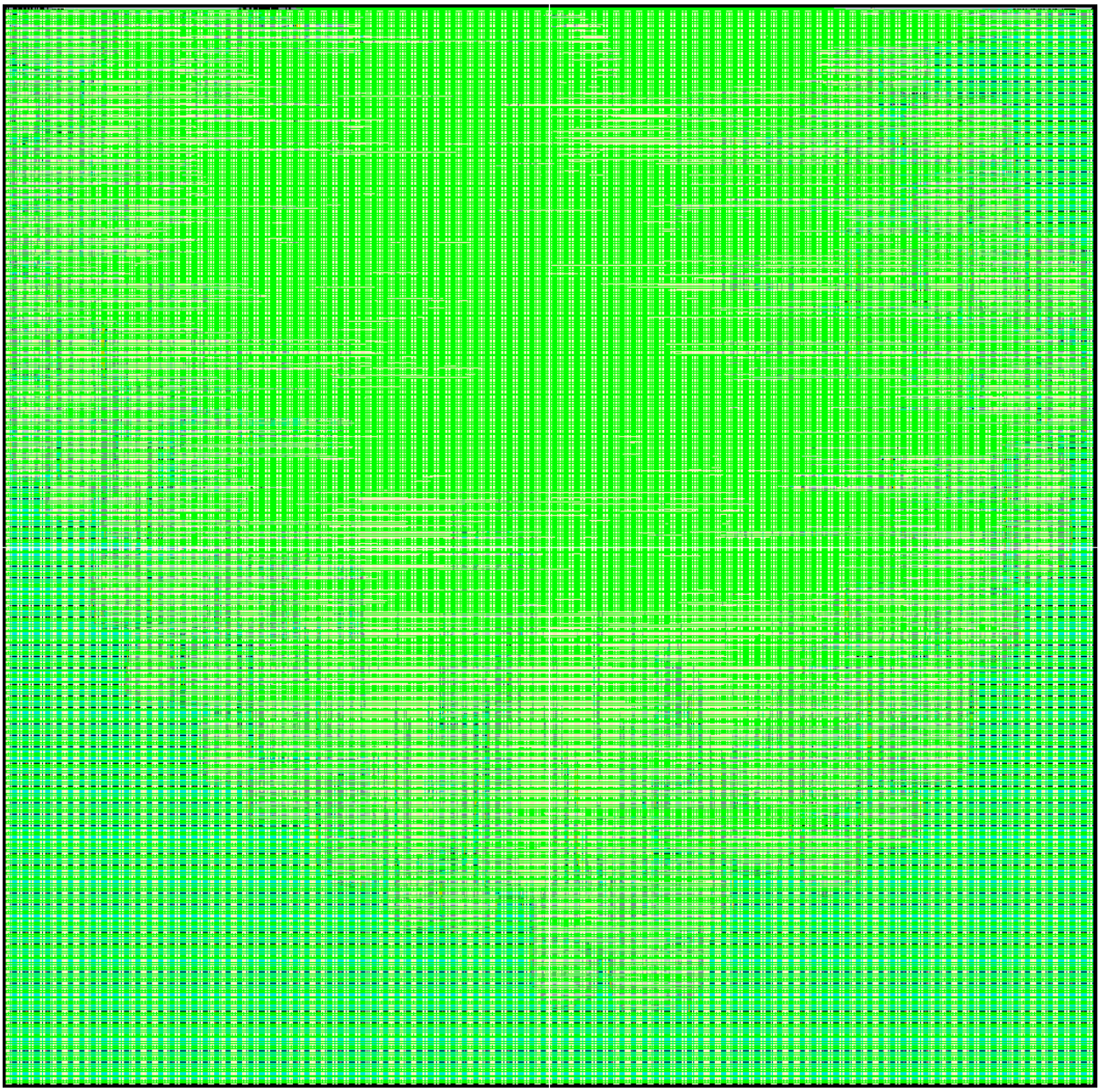

Figure 76: Optimized High-performance AES Core Layout

Table 50: Synthesized vs. Optimized High-performance AES Core Metrics

	Synthesized	Optimized	Percent Improvement
Average Frequency (MHz)	6.7	11.1	65.1%
Average Active Energy per Operation (pJ)	8995.6	10500.8	-16.7%
EDP (pJ*ns)	1335578	944078	29.3%
Leakage Power (µW)	1332.96	1679.81	-26.0%
Area (µm²)	1140041	1175599	-3.1%

Table 50 enables a comparison between the *synthesized* and *optimized high-performance* AES-256 cores. To reiterate, both circuits are sourced from the exact same gate-level netlist; therefore, their differences arise out of physical implementation alone. The application of timing constraints to the AES core, rather than applying blanket max transition constraints, boosts the performance by 65.1%. As demonstrated by the breakdown provided in Table 51, the delays through the key expansion and round logic dropped by 46% and 42%, respectively, aligning with the overall circuit performance improvement. Table 52 summarizes the primary sources of additional delay. The *synthesized high-performance* AES-256 core exemplifies that using max transition constraints during physical implementation is not a viable approach to generating *high-performance* circuits. The proportions of buffer delay through the key expansion and round logic for the *optimized* design are 18% and 39%, respectively, whereas these proportions spike to 59% and 61% in the *synthesized* design. Therefore, considering the *optimized* design to be the ideal case due to its comparatively high performance, the *synthesized* design exhibits excessive buffering in the datapath.

Table 51: Synthesized vs. Optimized High-performance AES Core DATA Cycle Breakdown

	Synthesized	Optimized
Key Expansion	23.7	12.6
Number of Iterations	1	1
Total Key Expansion	23.7	12.6
Round Logic	10.6	6.2
Number of Iterations	14	14
Total Round Logic	148.4	86.1
Total Cycle Delay	172.0	98.8

Table 52: Synthesized vs. Optimized High-performance AES Core Buffer Delay

	Synthesized		Optimized	
	Key Expansion	**Round Logic**	**Key Expansion**	**Round Logic**
Buffer Delay	13.9	90.6	2.3	33.6
Total Delay	23.7	148.4	12.6	86.1
Percentage	59%	61%	18%	39%

A summary of each circuit's composition is provided via Table 49 and Table 53. The instance count of all three upsized MTNCL cell categories decreased, with a 3× increase in minimum-sized MTNCL cells. This overall downsizing would improve both active and static energy efficiency; however, the total driver area increased by 32.7% from *128,304 µm²* to *170,316 µm²*. In addition to worsening active energy per operation, given the low-resistance nature of driver cells, this increase accounts for the 26% jump in static power dissipation. In pursuit of the lowest delay possible along the performance-related paths, Innovus undoubtedly prioritized lower parasitic routing delay over lower parasitic routing capacitance and, consequently, active energy per operation. Nonetheless, the performance increase overcomes the active energy degradation, resulting in a 29.3% reduction in EDP.

Table 49: Synthesized High-performance AES Core Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	8898	52918	0.5	0.5	13229
MTNCL X0P7	39070	205811	0.7	0.5	72034
MTNCL X1	56930	290936	1	0.5	145468
MTNCL X2	66775	461988	2	0.5	461988
BUF	51691	110229	1	1	110229
INV	10203	18075	1	1	18075
Total	233567	1139956	0	0	821023

Table 53: Optimized High-performance AES Core Composition and Switching Activity

Cell Category	Number Instances	Total Area	Size Factor	Activity Factor	Total Switching
MTNCL X0P5	35703	187524	0.5	0.5	46881
MTNCL X0P7	25656	131589	0.7	0.5	46056
MTNCL X1	48331	252332	1	0.5	126166
MTNCL X2	61983	433753	2	0.5	433753
BUF	41256	163180	1	1	163180
INV	2227	7136	1	1	7136
Total	215156	1175515	0	0	823173

5.3.3.4 Summary

Table 54: Structural vs. Optimized High-performance AES Core Metrics

	Structural	Optimized	Percent Improvement
Average Frequency (MHz)	6.6	11.1	69.3%
Average Active Energy per Operation (pJ)	5898.5	10500.8	-78.0%
EDP (pJ*ns)	897606	944078	-5.2%
Leakage Power (µW)	410.57	1679.81	-309.1%
Area (µm²)	301031	1175599	-290.5%

Table 54 encompasses an overview of the design QoR changes accompanying a transition from *structural* design to a high-quality synthesis flow and timing constraints during physical implementation. Given that the target application of this set of AES cores is *high-performance*, the 69.3% increase in throughput is a substantial enhancement. However, the designer must consider the tradeoffs associated with the *optimized* design. While the active energy scales alongside performance, the leakage power and area of the *optimized* design are around three times greater than those of the *structural* design. Thus, for every percentage of increased performance, both leakage power and area degrade by 4.46%, an exceedingly expensive tradeoff. If throughput is the primary concern for the target use case, the *optimized* design is superior.

A keen reader might assert that two instances of the *structural* design operating in parallel would exceed the performance of the *optimized* design at 19% greater throughput, 12% higher active energy, and 50% less leakage power and area—an ostensibly superior tradeoff in design metrics. However, if the *optimized*, *low-power* AES-256 core were instantiated multiple times, it would greatly exceed any of the other designs in terms of active and static energy efficiency. Furthermore, this core would achieve the highest performance density if provided tighter timing constraints during physical implementation. Finally, the analysis thus far has focused exclusively on design QoR; the *optimized* designs are the only implementations that can be trusted to function reliably due to their safeguards against the timing race conditions presented in Section 2.5.

5.4 Performance Estimation

Table 55: Predicted vs. Measured Optimized Design Throughput

	Predicted	Measured	Improvement
LP Adder *	73.3	351.7	480%
HS Adder *	701.8	592.4	84%
1-stage LP Montgomery *	39.8	108.8	274%
4-stage HS Montgomery	213.7	252.0	118%
8-stage HS Montgomery	298.5	306.3	103%
LP AES-256	2.5	4.7	191%
HS AES-256	8.4	11.1	132%

This section has been included to evaluate one of this work's key claims: the application of timing constraints during physical implementation enables the circuit designer to estimate the performance of the target MTNCL circuit prior to layout completion. Table 55 facilitates this evaluation by presenting the predicted throughput based on the timing constraints, the measured throughput, and a proportion of the two for each of the *optimized* designs. As demonstrated by the generalized version of Equation 2 from Section 3.3.2 (added below for reference), the MTNCL circuit cycle time is proportional to the *I path* delay, with an additional *TH22* delay set to *150 ps* for the table. The variable *n* represents the number of *I cycles* per operation. Both adders and the *low-power* Montgomery modular multiplier are a single stage, so *n* was set to 1, indicated by the (*) in the table. The *high-performance* Montgomery modular multipliers utilized the standard *n* of 2. The AES-256 cores required a slightly more complex calculation, including one *I path* delay through the key expansion and 14 *I path* delays through both the round logic and the two adjacent ring registers.

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

Depending on the circuit in question, there are potentially several reasons why the predicted performance diverged from the measured value. First, while the *I paths* correspond to the *worst-case* delay, MTNCL circuits achieve *average-case* delay through completion detection, and the magnitude of this divergence depends on the circuit architecture. For instance, the *low-power* adder and *low-power* Montgomery modular multiplier employ RCAs, where the *average-case* delay is substantially less than the *worst-case* across the entire carry chain. Second, the path delays from Innovus utilize a PVT corner with 10% IR drop, but the actual IR drop in the MTNCL circuit is variable, averaging less than 10%. Third, the designs often exhibit some positive slack on the *I paths*, meaning the actual path delay is less than the constraint value. The *low-power* designs, in particular, were given very loose constraints to avoid optimizing for performance, which further exacerbates this effect.

The factors presented thus far explain why six of the seven *optimized* circuits exceed the predicted performance in transistor-level simulation. The *high-performance* adder, on the other hand, operated at only 84% of the predicted performance. While all the 1-stage circuits utilized an *n* value of one, as described in Section 5.1.3.2, the overall circuit critical path for these designs encompasses the I/O-related paths, which introduce additional delay that is not included in the equation. The *low-power* adder and Montgomery modular multiplier are also 1-stage, but their *average-case* performance overcomes the additional I/O delay and results in them exceeding the projected throughput.

In summary, precisely predicting performance for MTNCL circuits during implementation using the synchronous *worst-case* timing analysis paradigm is difficult and not reliable for *1-stage circuits*. However, the equation and *I path* delays can yield approximate performance metrics, with the expectation that the final multi-stage MTNCL circuit should consistently

outperform the projected throughput. Measuring *average* performance with greater precision will require transistor-level or SDF-annotated logical simulations after physical implementation and parasitic extraction.

5.5 Reliability

This section briefly discusses some reliability-related observations from the evaluation phase of this work. One of the core assertions of this dissertation is that the application of timing constraints to MTNCL circuit design ensures these circuits will operate reliably in the specified PVT corners. As explained previously, the input interface of MTNCL circuits is especially vulnerable to the timing race conditions outlined in Section 2.5. Thus, the circuit's functionality—whether it operates correctly, produces incorrect results, or completely deadlocks—often hinges on how quickly the surrounding environment responds to requests for DATA and NULL via *Ko*. This is demonstrated by the common practice of delaying the assignment of DATA and NULL waves to the input of MTNCL circuits during simulation, where the testbench would otherwise respond to requests instantaneously. Although this lack of delay is unrealistic in a physical circuit, one of the key claims and proposed benefits of asynchronous circuits is their improved reliably in the presence of delay variability compared to synchronous alternatives. The circuit functionality should most certainly not hinge on the delays at the I/O interfaces; however, this is often the case for unconstrained MTNCL circuits.

While simulating the circuits used for the evaluation of this work, the required minimum delay between requests via *Ko* and assignment of DATA and NULL to the input interface for each of the circuits was measured with a resolution of *100 ps*. The collected values are listed in Table 56. Simulating any of the designs with a delay *100 ps* less than the value indicated in the

table results in functional failure, mostly due to deadlock from the DATA handshaking race condition.

The key observation is that almost all the *structural* and *synthesized* circuits required a *non-zero* delay for correct functionality. Conversely, *none* of the *optimized* circuits required any delay. This measurement demonstrates that timing closure using the developed constraints yields fully modular MTNCL circuits that can be seamlessly integrated into other circuits without any boundary timing issues when operating at the evaluated PVT corners.

Table 56: Required Testbench Assignment Delays

	Circuit	Minimum Delay (ps)
Structural	LP Adder	100
	HS Adder	100
	LP 1-Stage Montgomery	300
	HS 4-stage Montgomery	200
	LP AES-256	300
	HS AES-256	200
Synthesized	LP Adder	0
	HS Adder	400
	LP 1-Stage Montgomery	700
	HS 4-stage Montgomery *	500
	HS 8-stage Montgomery *	100
	LP AES-256	600
	HS AES-256	800
Optimized	LP Adder	0
	HS Adder	0
	LP 1-Stage Montgomery	0
	HS 4-stage Montgomery	0
	HS 8-stage Montgomery	0
	LP AES-256	0
	HS AES-256	0

Recall from Section 2.5 that *every* stage of an MTNCL circuit is vulnerable to *all* four of the timing race conditions and could trigger functional failure. While iterating on the evaluation circuits, both *synthesized high-performance* Montgomery modular multipliers experienced the DATA handshaking hazard. Alarmingly, this race condition did *not* occur at the input interface but *within* the circuit, resulting in deadlock during simulation. Like a hold violation, this issue can be practically *unsolvable* post-silicon. Prior to successful simulation and data collection, these circuits required multiple ECO iterations, where delay cells were added to the *Ko* signals. It is important to note that although the post-ECO circuits functioned in the simulations conducted for this work, they remain susceptible to this race condition—or any of the other three—if subjected to different PVT corners or data patterns during simulation or following fabrication. Therefore, achieving timing closure with the developed constraints is the *only* viable solution to mitigate the race conditions inherent to MTNCL.