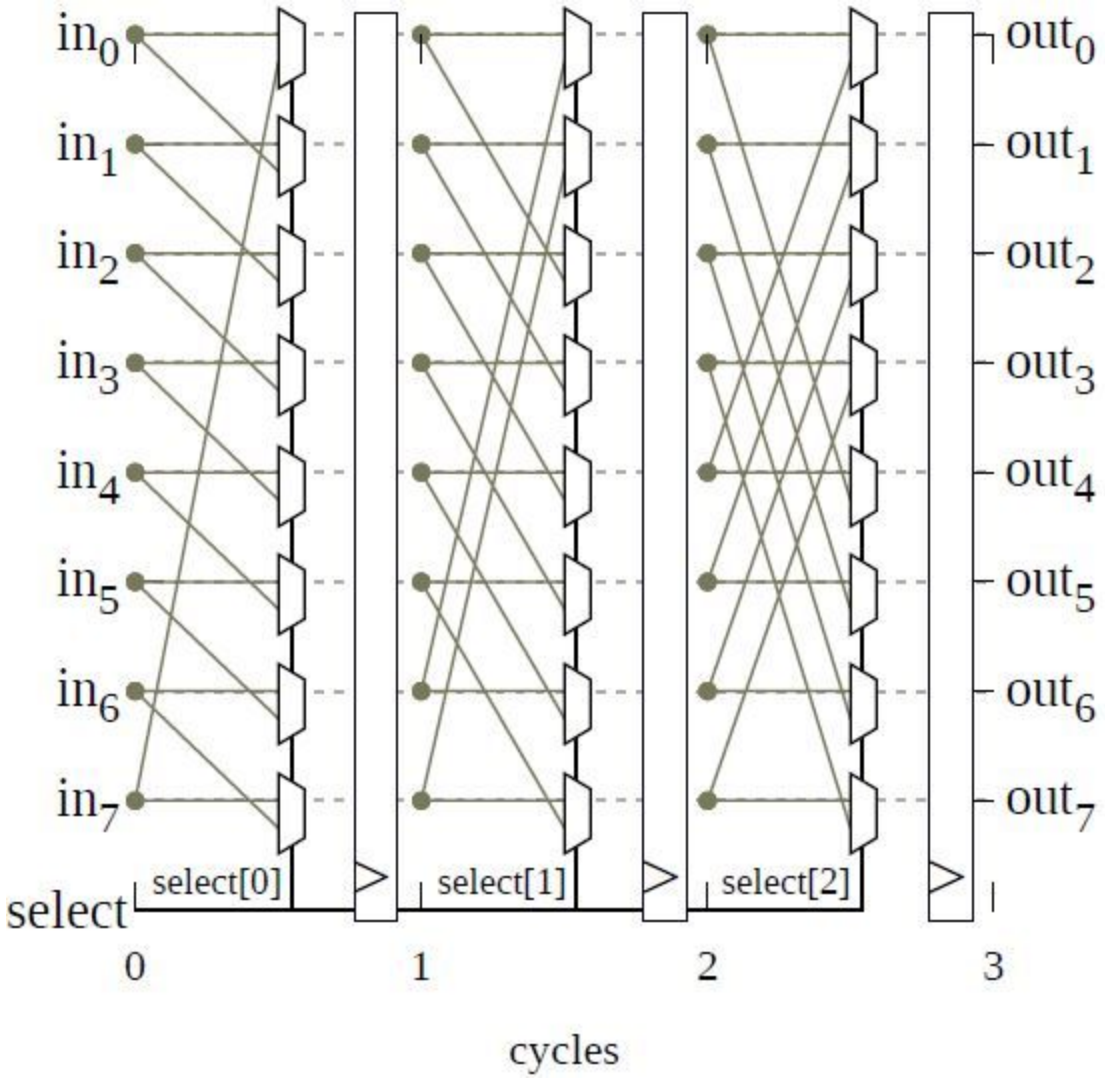
EE 599: Assignment 2

Suraj Chakravarthi Raja | surajcha@usc.edu | USC ID: 1389-2888-44

GitHub Repo: github.com/solaremperor/EE-599_SurajChakravarthiRaja_1389288844

Problem 1: Barrel Shifter



The barrel shifter is a pipelined bit-shifter capable of rotating an 'n' bit number by upto '$n-1$' bits. It uses 'n' multiplexers (MUXes) at every one of $log_2 n$ stages.
Here are the specifications of the sorting engine, for sorting a sequence of 'n' elements.

- Number of multiplexers per stage: n
- Number of clocks to produce a result: $log_2 n$ clock cycles
- This design is pipelined
- Total number of MUXes in barrel shifter: $n \bullet log_2 n$ clock cycles
- The design uses a butterfly network design.

Design files

The complete set of design files are available as in the GitHub repo.

- **barrel_shifter_top.sv** : This top-level module contains the complete RTL design of barrel shifter

```
module barrel_shifter_top #(
        parameter   BIT_WIDTH   = 8,
        parameter   STAGE_WIDTH = 16,
        localparam  STAGE_DEPTH = $clog2(STAGE_WIDTH)
)
(
        input                       clk,
        input                       reset,
        input                       valid_in,
        input   [  BIT_WIDTH-1 : 0]   in  [0 : STAGE_WIDTH-1],
        input   [STAGE_DEPTH-1 : 0]   sel,
        output                      valid_out,
        output  [  BIT_WIDTH-1 : 0]   out [0 : STAGE_WIDTH-1]
);
```

  - The parameter `BIT_WIDTH` sets the bit-width of each element. It is set to 8 bits for this report.
  - The parameter `STAGE_WIDTH` is used to set the number of elements in the input array to be sorted. *NOTE: Remember that this number must always be a power of 2.*
  - When the `reset` is cleared, the barrel shifter starts its operation, performing one shifting operation in every clock.
  - The `sel` bus is STAGE_DEPTH= $log_2 n$ bits wide and is used by the barrel shifter to rotate the data elements by 0 - $(n-1)$ positions.
  - The corresponding `valid_in` input line is set to ACTIVE HIGH along with the input data to denote that the data being ingested is valid. This valid bit travels through the pipeline along with the data, through to `valid_out`.
  - So every output data in the `out` bus with an ACTIVE HIGH `valid_out` is considered a valid array data that has been shifted.

- **butterfly_mux_stage.sv** : This a combinational design of a single stage of the barrel shifter.

```
module butterfly_mux_stage #(
        parameter BIT_WIDTH       = 8,
        parameter STAGE_WIDTH     = 16,
        parameter STAGE_POS       = 1,
        localparam STAGE_ROT      = (2**STAGE_POS)
)
(
        input   [BIT_WIDTH-1 : 0]   in  [0 : STAGE_WIDTH-1],
        input                       sel,
        output  [BIT_WIDTH-1 : 0]   out [0 : STAGE_WIDTH-1]
);
```

  - The parameter `BIT_WIDTH` sets the bit-width of each element. It is set to 8 bits for this report.
  - The parameter `STAGE_WIDTH` is used to set the number of elements in the input array to be sorted. This is also the number of 2-way multiplexers that are instantiated in the design.
    - *NOTE: Remember that this number must always be a power of 2.*

- The parameter `STAGE_POS` is used to define the position of the butterfly MUX stage within the pipelined barrel shifter. This defines the connectivity of the second input to each MUX. The second input of the i[th] connects to the $(i - 2^{STAGE\_POS})^{th}$ input data element (negative values wrap around) to form the butterfly connectivity structure.
- The `sel` input line connects to the select line of every MUX, and defines which of the MUX inputs is sent to the output bus of the MUX.
- We will need a total of $log_2 n$ of these shuffling butterfly MUX stages to build the complete barrel shifter.

- **mux_2way.v** : This a combinational design of a single two-way multiplexer (MUX).

```
module mux_2way #(
        parameter BIT_WIDTH = 8
)
(
        input   [BIT_WIDTH-1 : 0]   in1,
        input   [BIT_WIDTH-1 : 0]   in2,
        input                       sel,
        output  [BIT_WIDTH-1 : 0]   out
);
```

  - The parameter `BIT_WIDTH` sets the bit-width of each 2-way MUX. It is set to 8 bits for this report.
  - The MUX has two input buses (`in1, in2`) that are `BIT_WIDTH` bits wide.
  - The sel input line selects one of these two inputs to be passed on to the output bus (`out`).
    - An ACTIVE LOW line will select `in1`.
    - An ACTIVE HIGH line will select `in2`.
  - The butterfly connectivity structure of each shuffling stage of the barrel shifter needs n of these MUXes to be built. There are $log_2 n$ such stages, yielding a total of $n \bullet log_2 n$ of these MUXes.

- **barrel_shifter_tb.sv** : Testbench used to simulate, probe and validate the design.

```
module barrel_shifter_tb #(
        parameter   BIT_WIDTH = 8,
        parameter   STAGE_WIDTH = 16,
        localparam  STAGE_DEPTH = $clog2(STAGE_WIDTH),
        localparam  NUM_INPUTS  = 3
)
(
);
```

  - The parameter `BIT_WIDTH` sets the bit-width of each element. It is set to 8 bits for this report.
  - The parameter `STAGE_WIDTH` is used to set the number of elements in the input array to be sorted. This is also the number of 2-way multiplexers that are instantiated in the design.
    - *NOTE: Remember that this number must always be a power of 2.*
  - The testbench randomly generates a sequence of '`STAGE_WIDTH`' number of unsigned integer elements for barrel-shifting/rotating.
  - Another unsigned random number (range: 0 to `STAGE_WIDTH`-1) is used as the number of positions by which the input data array is rotated to the output data array.
  - The output is read when the `valid_out` line goes ACTIVE HIGH in $log_2 n$ clock cycles.

Simulation waveforms (16 element sequence)

The simulation uses a 10 ns clock (`clk`). The reset line is cleared at the 15 ns mark.

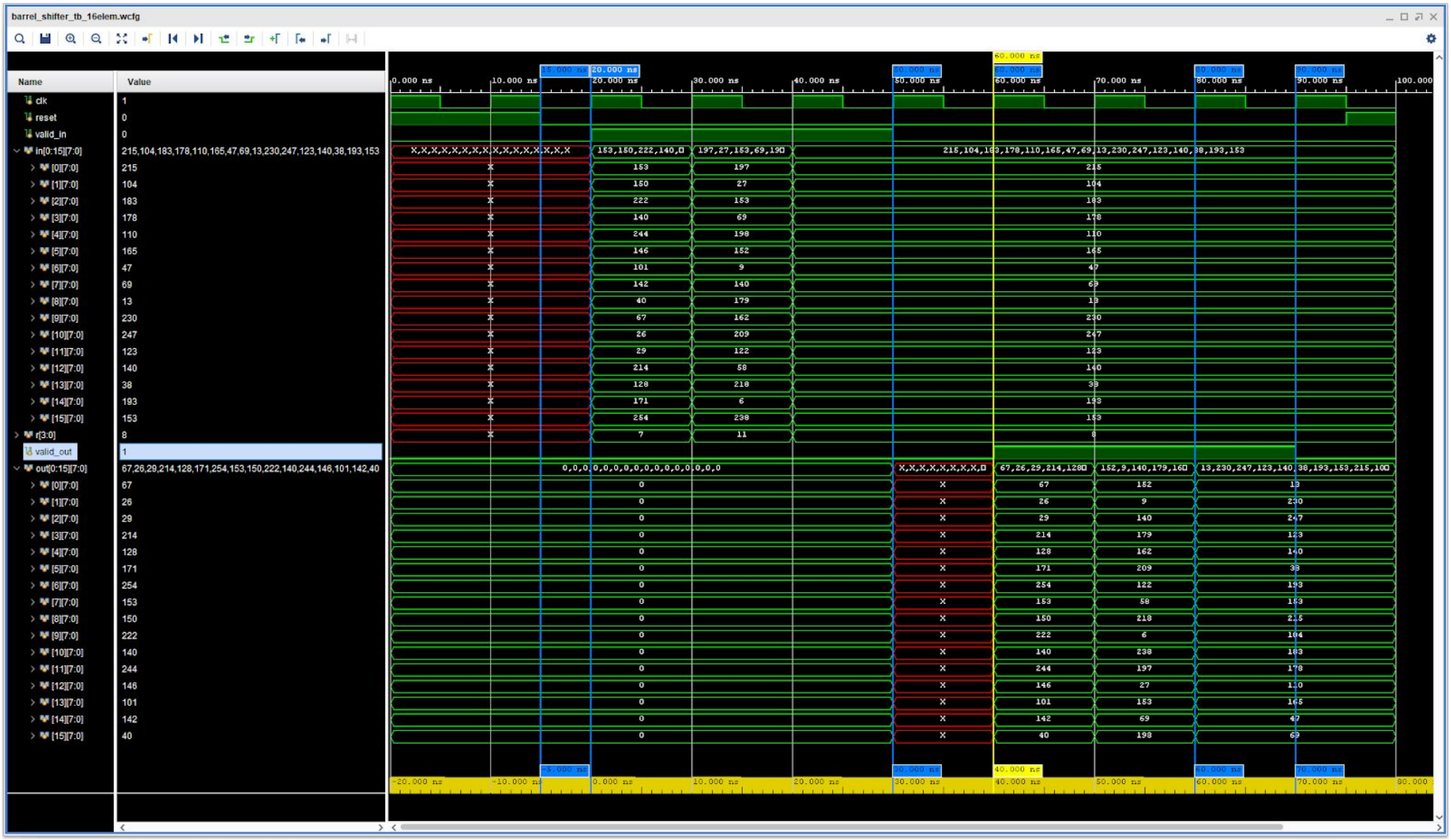
In the simulation waveform, you will see that at the 20 ns mark, the `valid_in` line is set to HIGH and the barrel shifting begins. The first input data array is ***153***,*150,222,140,244,146,101,142,40,67,26,29,214,128,171,254* which will be rotated right by by 7 (bus '`r`' in the simulation waveform) places.

After 4 clock cycles (at the 60 ns mark), the `valid_out` line is set to HIGH and the '`out`' bus shows the result is set to the rotated values. As expected, the output is `67,26,29,214,128,171,254,`**`153`**`,150,222,140,244,146,101,142,40`.

The first element from the input is emphasized in read to showcase that, indeed, the output is rotated right by 7 places.

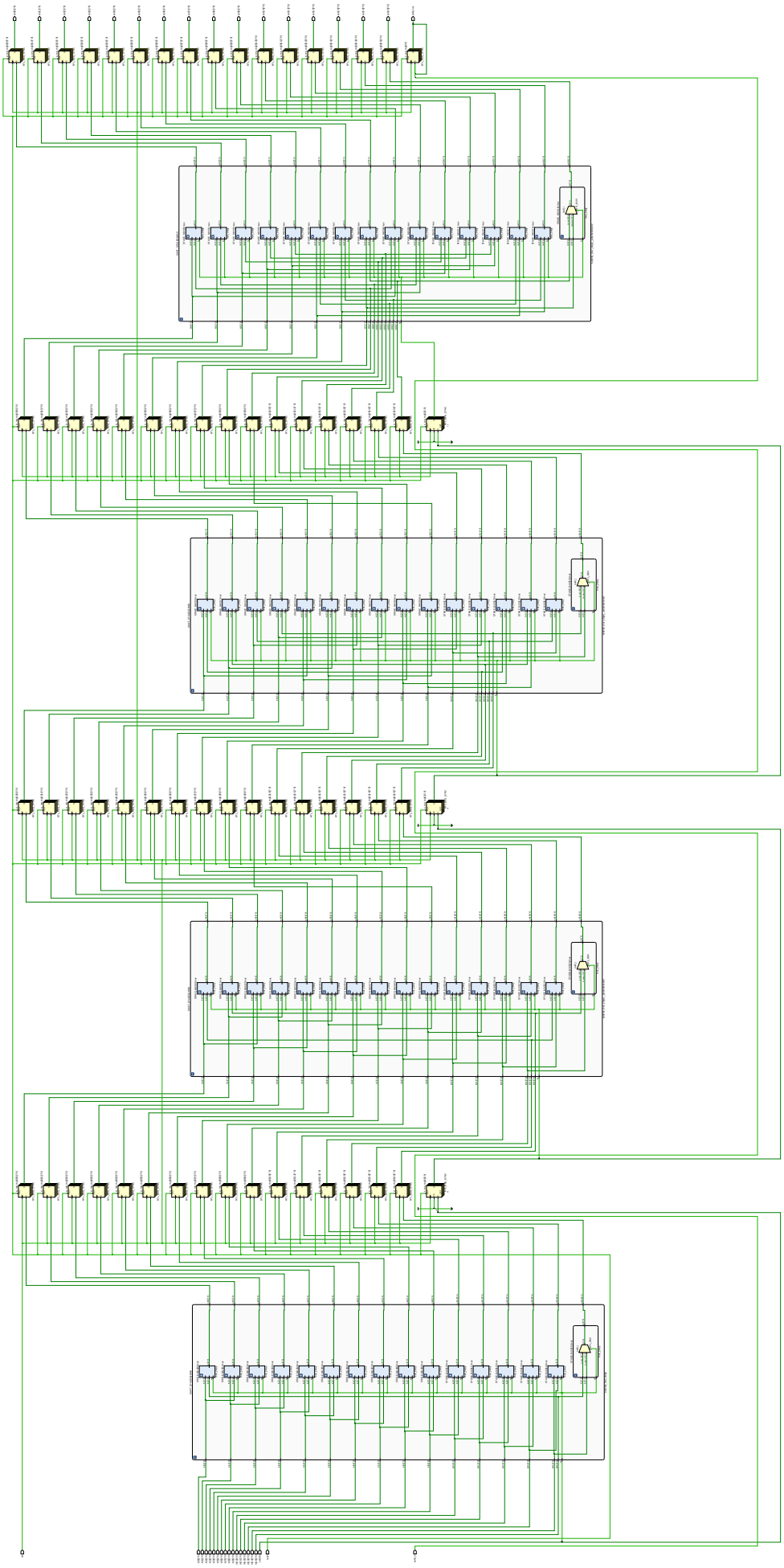
The waveform figure also breaks out the input and output busses into their constituent elements to prove that the barrel shifter is indeed pipelined. This waveform figure looks at three sets of randomized inputs being rotated. The highlighted time point (60 ns mark - see value column for actual values at this mark) shows the output for the first data array in the '`out`' bus. At the same time, the last input that was sent in the '`in`' bus is also shown.

(The enlarged waveform figures is on the next page)


barrel_shifter_tb_16elem.wcfg
Name
Value
clk
1
reset
0
valid_in
0
in[0:15][7:0]
215,104,183,178,110,165,47,69,13,230,247,123,140,38,193,153
[0][7:0]
215
[1][7:0]
104
[2][7:0]
183
[3][7:0]
178
[4][7:0]
110
[5][7:0]
165
[6][7:0]
47
[7][7:0]
69
[8][7:0]
13
[9][7:0]
230
[10][7:0]
247
[11][7:0]
123
[12][7:0]
140
[13][7:0]
38
[14][7:0]
193
[15][7:0]
153
r[3:0]
8
valid_out
1
out[0:15][7:0]
67,26,29,214,128,171,254,153,150,222,140,244,146,101,142,40
[0][7:0]
67
[1][7:0]
26
[2][7:0]
29
[3][7:0]
214
[4][7:0]
128
[5][7:0]
171
[6][7:0]
254
[7][7:0]
153
[8][7:0]
150
[9][7:0]
222
[10][7:0]
140
[11][7:0]
244
[12][7:0]
146
[13][7:0]
101
[14][7:0]
142
[15][7:0]
40
0.000 ns
10.000 ns
20.000 ns
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 ns
80.000 ns
90.000 ns
100.000
-20.000 ns
-10.000 ns
0.000 ns
10.000 ns
20.000 ns
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 ns
80.000

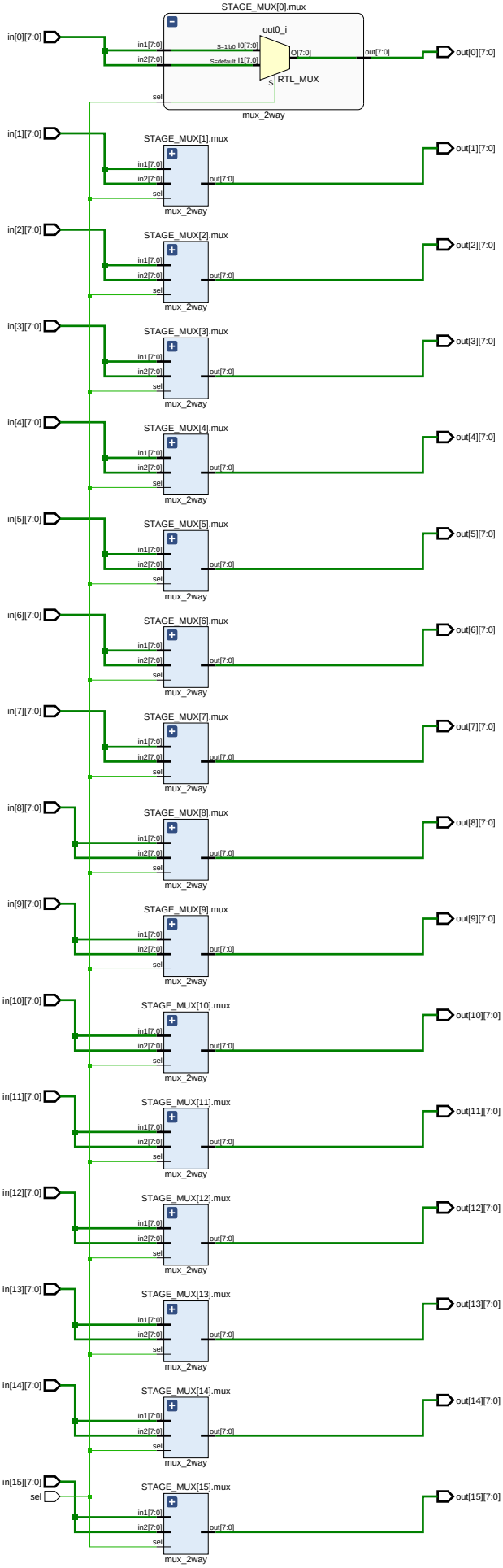
Elaborated Design (16 element sequence - full barrel shifter)

(Inserted in the next page - for brevity only one MUX expanded in each stage)



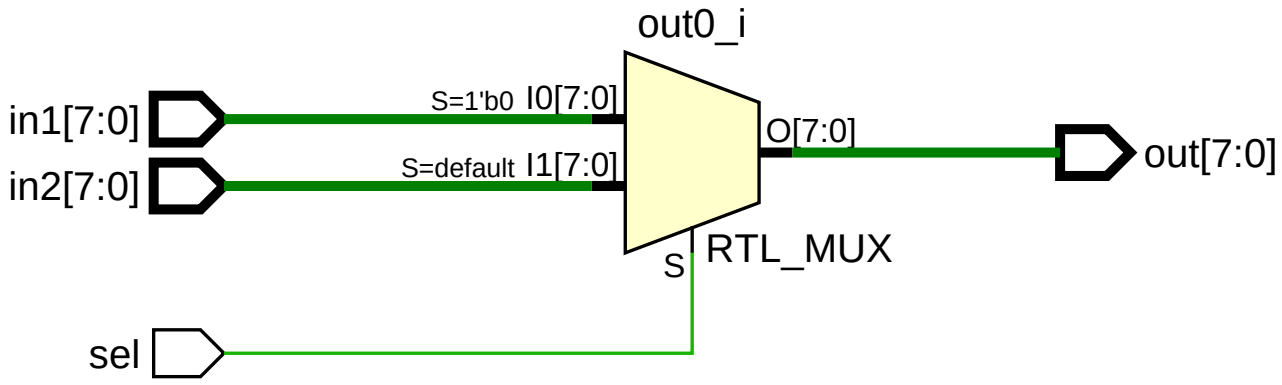
Elaborated Design (16 element sequence - butterfly_mux_stage)

(Inserted in the next page - for brevity only one MUX expanded. This is the elaborated design for **stage 4** butterfly mux shuffler)


STAGE_MUX[0].mux
out0_i
in1[7:0]
in2[7:0]
S=1'b0 I0[7:0]
S=default I1[7:0]
O[7:0]
S
RTL_MUX
out[7:0]
sel
mux_2way
in[0][7:0]
out[0][7:0]
STAGE_MUX[1].mux
in[1][7:0]
out[1][7:0]
STAGE_MUX[2].mux
in[2][7:0]
out[2][7:0]
STAGE_MUX[3].mux
in[3][7:0]
out[3][7:0]
STAGE_MUX[4].mux
in[4][7:0]
out[4][7:0]
STAGE_MUX[5].mux
in[5][7:0]
out[5][7:0]
STAGE_MUX[6].mux
in[6][7:0]
out[6][7:0]
STAGE_MUX[7].mux
in[7][7:0]
out[7][7:0]
STAGE_MUX[8].mux
in[8][7:0]
out[8][7:0]
STAGE_MUX[9].mux
in[9][7:0]
out[9][7:0]
STAGE_MUX[10].mux
in[10][7:0]
out[10][7:0]
STAGE_MUX[11].mux
in[11][7:0]
out[11][7:0]
STAGE_MUX[12].mux
in[12][7:0]
out[12][7:0]
STAGE_MUX[13].mux
in[13][7:0]
out[13][7:0]
STAGE_MUX[14].mux
in[14][7:0]
out[14][7:0]
STAGE_MUX[15].mux
in[15][7:0]
sel
out[15][7:0]

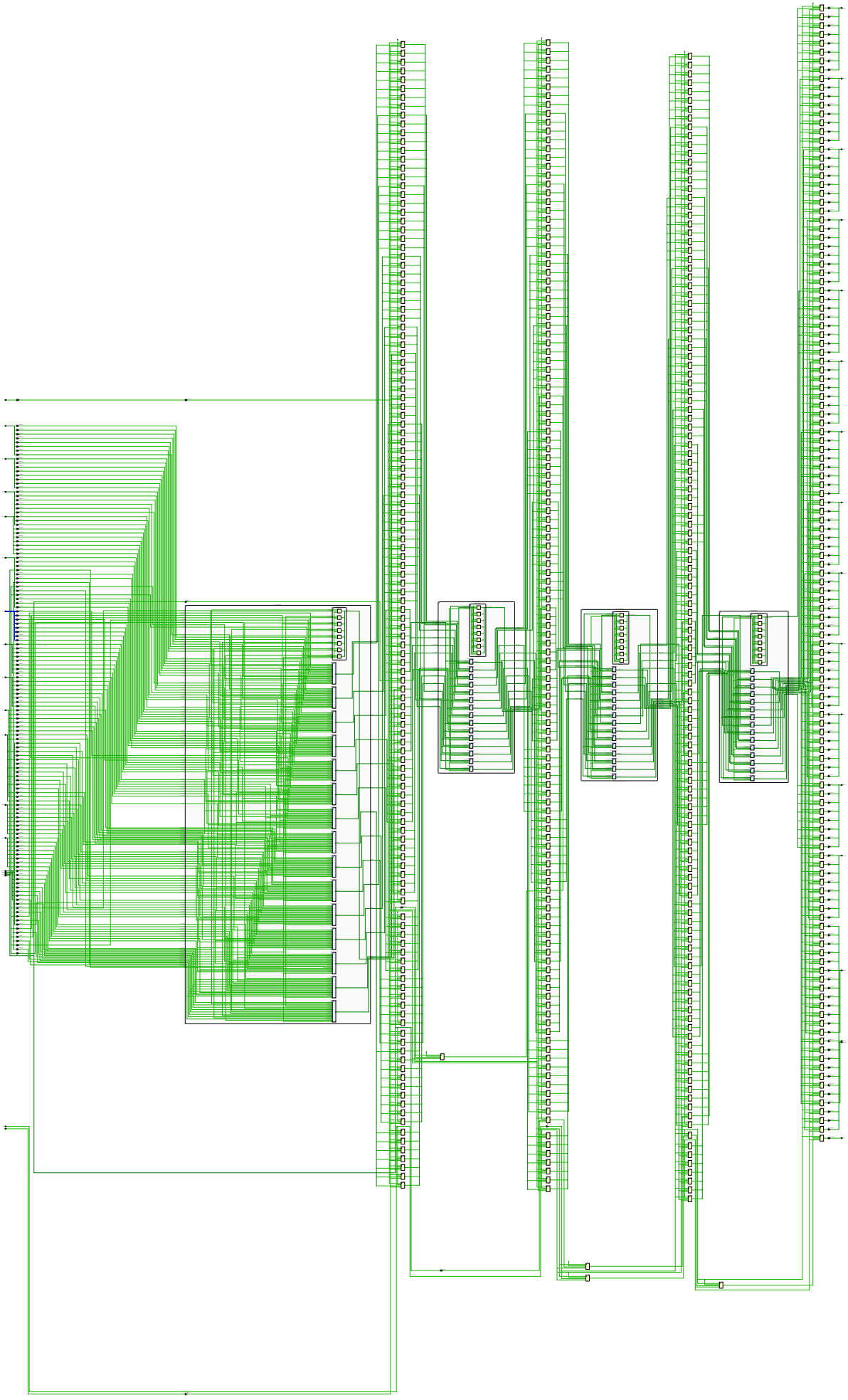
Elaborated Design (16 element sequence - 2-way MUX)

(Inserted in the next page)


out0_i
in1[7:0]
in2[7:0]
S=1'b0
I0[7:0]
S=default
I1[7:0]
O[7:0]
out[7:0]
S
RTL_MUX
sel

Synthesized Schematic (16 element sequence)

(Inserted in the next page - for brevity, only one MUX expanded in each shuffling stage <butterfly_mux_stage>)

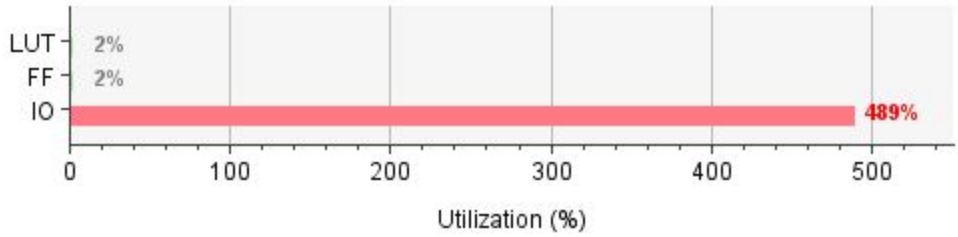


Synthesis Reports (16 element sequence)

Resource utilization report (16 element sequence)

Summary

Resource	Utilization	Available	Utilization %
LUT	256	14400	1.78
FF	522	28800	1.81
IO	264	54	488.89



Timing estimation report (16 element sequence)

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8.609 ns	Worst Hold Slack (WHS):	0.140 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	390	Total Number of Endpoints:	390	Total Number of Endpoints:	523

All user specified timing constraints are met.

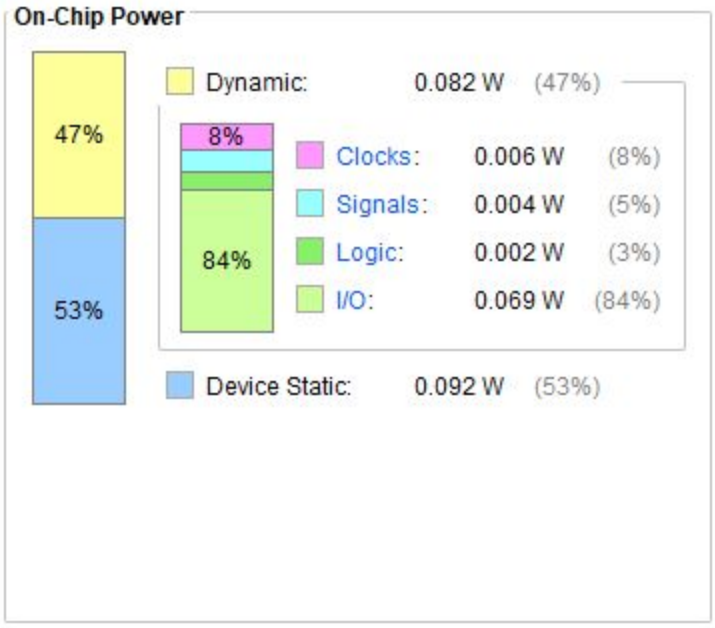
Power estimation report (16 element sequence)

Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

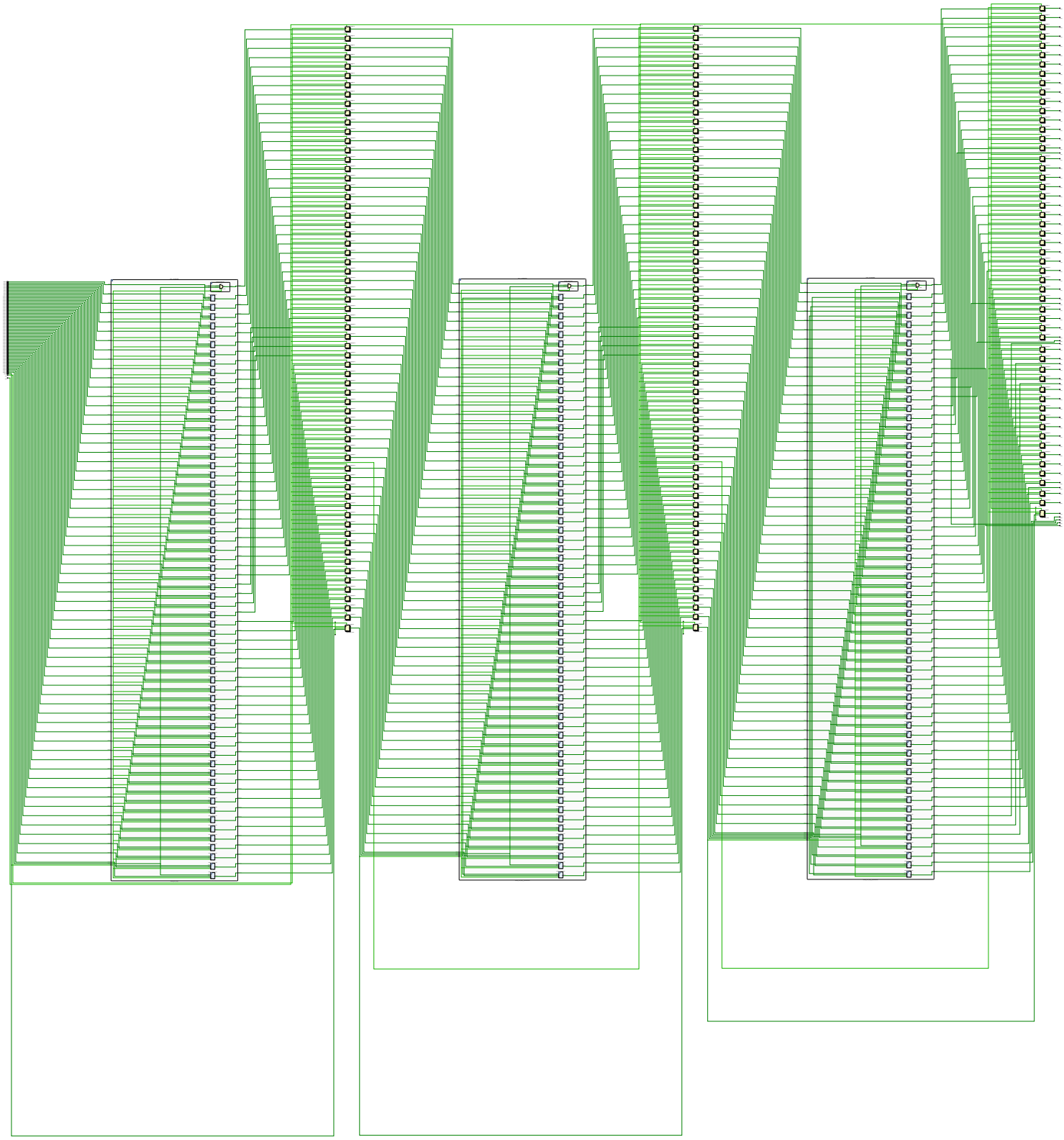
Total On-Chip Power:	**0.174 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**27.0°C**
Thermal Margin:	73.0°C (6.1 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

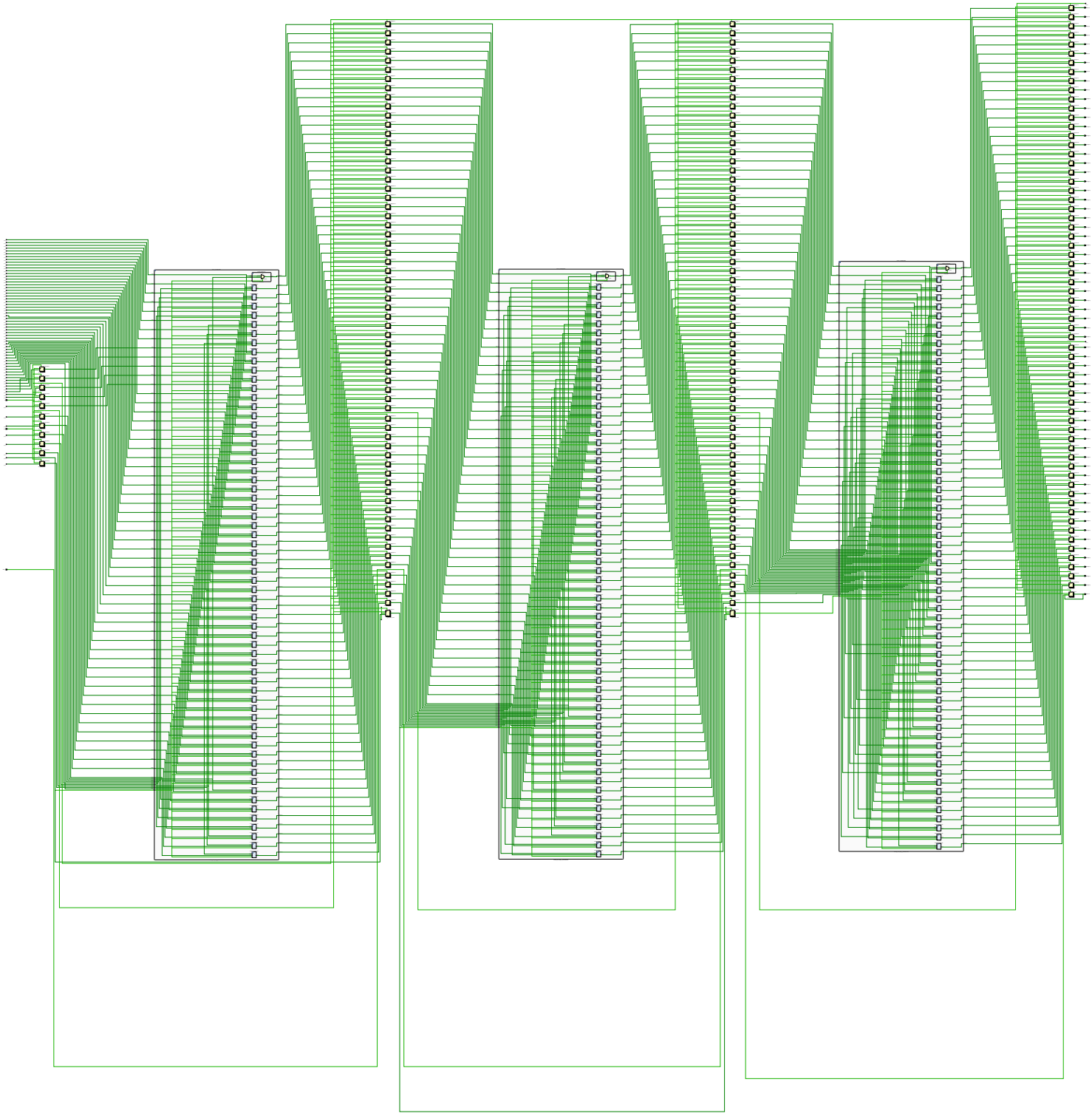
Launch Power Constraint Advisor to find and fix invalid switching activity



Elaborated Design (64 element sequence - full barrel shifter)

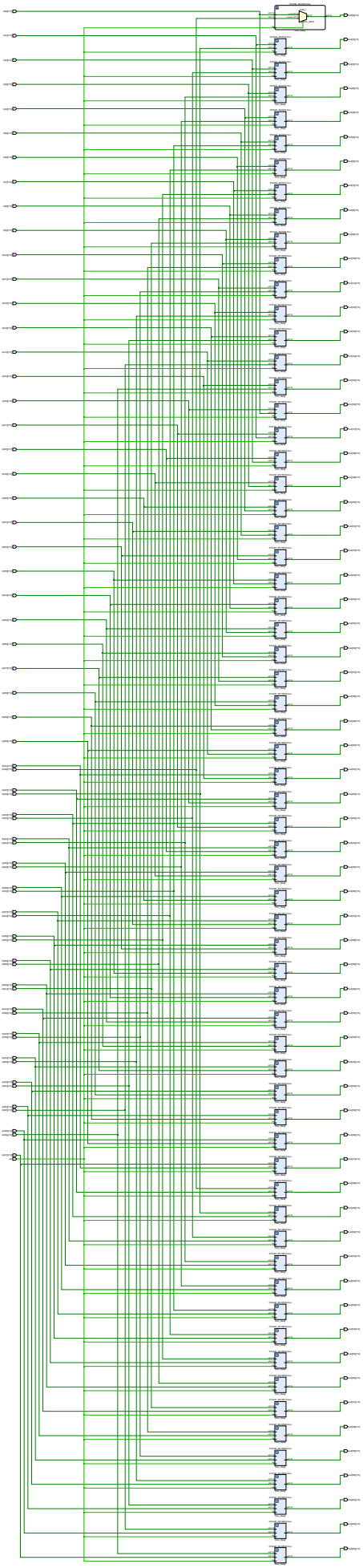
(Inserted in the next page - for brevity only one MUX expanded in each stage)





Elaborated Design (64 element sequence - butterfly_mux_stage)

(Inserted in the next page - for brevity only one MUX expanded. This is the elaborated design for **stage 4** butterfly mux shuffler)



Elaborated Design (64 element sequence - 2-way MUX)

(Not included for brevity since it remains unchanged from 16 element sequence design)

Synthesized Schematic (64 element sequence)

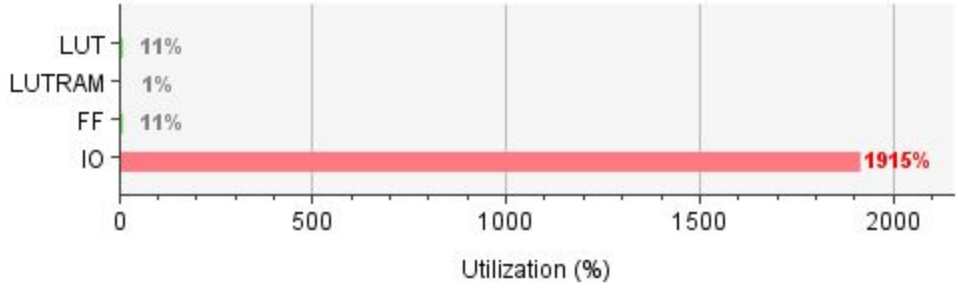
(Schematic is 23 pages long - inserted in **Appendix A**)

Synthesis Reports (64 element sequence)

Resource utilization report (64 element sequence)

Summary

Resource	Utilization	Available	Utilization %
LUT	1542	14400	10.71
LUTRAM	3	6000	0.05
FF	3098	28800	10.76
IO	1034	54	1914.81



Timing estimation report (64 element sequence)

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8.565 ns	Worst Hold Slack (WHS):	0.122 ns	Worst Pulse Width Slack (WPWS):	4.146 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	2579	Total Number of Endpoints:	2579	Total Number of Endpoints:	3102

All user specified timing constraints are met.

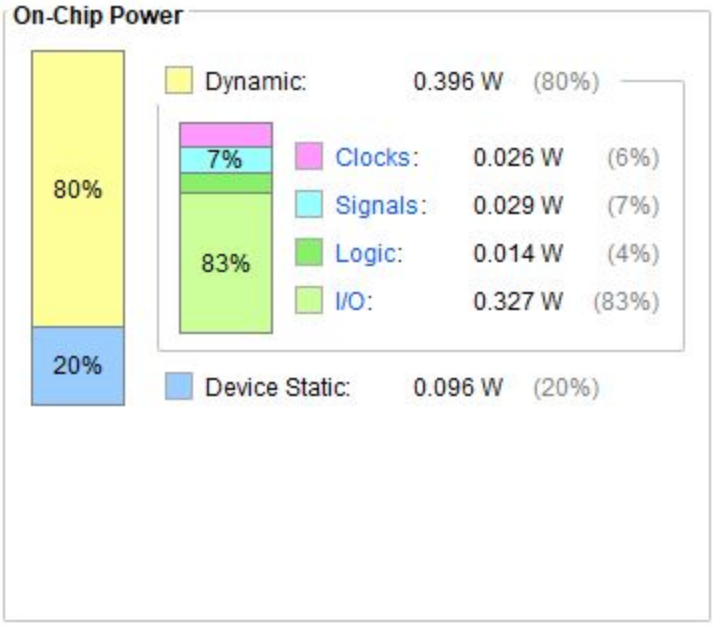
Power estimation report (64 element sequence)

Summary

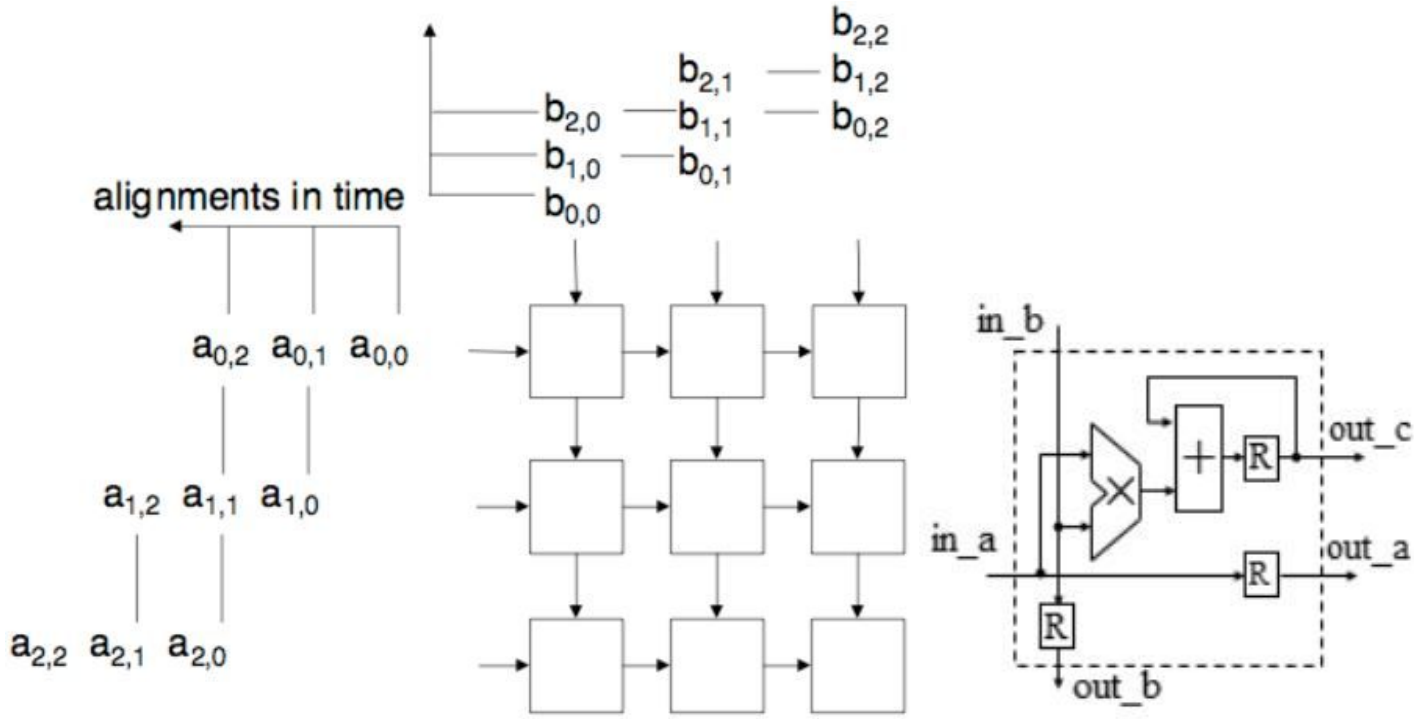
Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.492 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**30.7°C**
Thermal Margin:	69.3°C (5.8 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



Problem 2: Systolic Array for Dense Matrix-Matrix Multiplication



Given two matrices A (size $m \times n$) and B (size $n \times p$), the product matrix C (size $m \times p$) is denoted as C = AB such that each element of C is computed as:

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j} \ \forall \ i \in \{1, 2, ..., m\}, \ j \ \varepsilon \{1, 2, ..., p\}$$

For our problem, we will have both A and B matrices be of size $n \times n \ such \ that \ n = 2^r$

We will use a systolic array to do this processing. There are $n \times n$ systolic cells (processing elements - PEs) in a systolic array. These PEs are arranged like an $n \times n$ matrix. Each PE has a multiply-accumulate (MAC) unit that accepts two input integers that are multiplied and accumulated when both the input enable lines are HIGH. When both input enables go HIGH an output valid line which goes HIGH to indicate that the PE has started its accumulation process. The accumulated output is the output C value.

In addition, each PE registers and passes the input A coming from the left (along with its 'enable' input line) to the right. Similarly, it also registers and passes the input A coming from the top (along with its 'enable' input line) to the bottom.

When $n \times n$ such PEs are arranged like a matrix, we get a systolic array where the final output of the (i, j)th PE is the (i, j)th element of the resultant product matrix 'C.'

The input matrices to the systolic array are split up and sent in the order depicted in the figure. On each side, the one array input is sent in at every clock cycle.

Design Problems

Let us consider a systolic array for $n \times n$ dense matrix multiplication.

- This design needs $n \times n$ systolic cells in total. Each will have an internal MAC unit.
- Inputs are fed at the left and top in a particular order.
- If we have k-bit inputs, we would need a 2k-bit output at the output of the multiply units. The accumulators, therefore, must also be 2k bits wide. So, the final result of each PE will be 2k bits wide.
- The **result appears at the output (*3n-2)* clock cycles after the first positive edge**. This is because each matrix is split into $(\#rows + \#cols - 1) = (2n - 1)$ input arrays. After the last line is ingested, it takes $(n - 1)$ clock cycles for the result to appear. So, the total is $3n - 2$ clock cycles.
- This design does NOT support pipelined multiplication of multiple matrices and needs to be reset before each matrix multiplication

Design files

The complete set of design files are available in the GitHub repo.

- **systolic_array_top.sv** : This top-level module contains the complete RTL design of systolic array.

```
module systolic_array_top #(
      parameter BIT_WIDTH = 8,
      parameter NUM_ROWS  = 16,
      parameter NUM_COLS  = NUM_ROWS
)
(
      input                              clk,
      input                              reset,
      input                              en_in_a,
      input   [   BIT_WIDTH -1 : 0]   in_a    [0 : NUM_ROWS-1],
      input                              en_in_b,
      input   [   BIT_WIDTH -1 : 0]   in_b    [0 : NUM_COLS-1],
      output  [(2*BIT_WIDTH)-1 : 0]   out_c   [0 : NUM_ROWS-1]    [0 :
NUM_COLS-1],
      output                             valid
);
```

  - The parameter `BIT_WIDTH` sets the bit-width of each input element. The output is twice that width.
  - The parameter `NUM_ROWS` defines the number of rows in the matrix multiplication.
  - The parameter `NUM_COLS` defines the number of columns in the matrix multiplication.
  - The systolics array must have square matrices, so `NUM_ROWS` and `NUM_COlS` are equal. For multiplying two matrices of size $n \times n$, they would be equal to n.
  - When the `reset` is cleared and the enable lines (`en_in_a` and `en_in_b`) of each input side (`in_a` and `in_b`) are set to HIGH, the processing begins at the positive edge of the clock cycle.
  - So long as `reset` is cleared, every input (with their enable lines set to HIGH) at the positive edge of the clock is considered valid for processing.
  - When the result is ready, the `valid` output line goes ACTIVE HIGH. The result is available in the `out_c` bus.

- **systolic_cell.sv** : This contains the RTL design of a single systolic cell, the building block of the systolic array.

```
module systolic_cell #(
      parameter BIT_WIDTH = 8
)
(
      input                              clk,
      input                              reset,
      input                              en_in_a,
      input   [   BIT_WIDTH -1 : 0]   in_a,
      input                              en_in_b,
      input   [   BIT_WIDTH -1 : 0]   in_b,
      output                             en_out_a,
      output  [   BIT_WIDTH -1 : 0]   out_a,
      output                             en_out_b,
      output  [   BIT_WIDTH -1 : 0]   out_b,
      output  [(2*BIT_WIDTH)-1 : 0]   out_c,
      output                             valid
);
```

  - The parameter `BIT_WIDTH` sets the bit-width of each input. It is set to 8 bits for this report.
  - When the `reset` is cleared and the enable lines (`en_in_a` and `en_in_b`) of each input side (`in_a` and `in_b`) are set to HIGH, the processing begins at the positive edge of the clock cycle.
  - So long as `reset` is cleared, every input (with their enable lines set to HIGH) at the positive edge of the clock is considered valid for processing.
  - The `valid` output line goes ACTIVE HIGH when the accumulator has started accumulating the result for the matrix multiplication.
  - This cell also registers and passes along the input data (`in_a` and `in_b`) along with their corresponding enable lines (`en_in_a` and `en_in_b`) as outputs (`out_a` & `out_b`, and `en_out_a` & `en_out_b` respectively).

- **mult_acc.sv** : This contains the RTL design of the basic multiply-accumulate (MAC) unit.

```
module mult_acc #(
      parameter BIT_WIDTH = 8
)
(
      input                              clk,
      input                              reset,
      input                              enable,
      input   [   BIT_WIDTH -1 : 0]   in1,
      input   [   BIT_WIDTH -1 : 0]   in2,
      output  [(2*BIT_WIDTH)-1 : 0]   out,
      output                             valid
);
```

  - The parameter `BIT_WIDTH` sets the bit-width of the adder unit.
  - As long as `reset` is cleared and `enable` is HIGH, the resultant ($out_{new} = (in1 \times in2) + out_{old}$) is computed and stored at the positive edge of the clock cycle.
  - The output (`out`) is the accumulated/stored value. Its bit-width is twice as wide as the input data.
  - So long as `reset` is cleared, every input at the positive edge of the clock is processed.

- The `valid` output line goes ACTIVE HIGH as soon as the MAC unit starts accumulating the product values.

- **systolic_array_tb.sv** : Testbench used to simulate, probe and validate the design.

```
module systolic_array_tb #(
      parameter   BIT_WIDTH = 8,
      parameter   NUM_ROWS  = 16,
      parameter   NUM_COLS  = NUM_ROWS,
      localparam  LINES   = NUM_ROWS + NUM_COLS - 1,
      localparam  NUM_RANGE = 10//(2**BIT_WIDTH)
)
(
);
```

  - The parameter BIT_WIDTH sets the bit-width of each input element. The output is twice that width.
  - The parameter NUM_ROWS defines the number of rows in the matrix multiplication.
  - The parameter NUM_COLS defines the number of columns in the matrix multiplication.
  - The testbench randomly generates A and B matrices of size '`NUM_ROWS x NUM_COLS`' filled with unsigned integer elements.
  - It then feeds the appropriate rows and columns of A and B respectively to the systolic array.
  - When the valid line of the systolic array goes HIGH, it obtains each of the n^2 elements of the resulting matrix out_c.

Simulation waveforms (16x16 matrices)

Randomly generated inputs:

A matrix:

[7] [4] [2] [2] [4] [7] [0] [5] [5] [5] [1] [3] [6] [6] [2] [8]
[5] [8] [1] [1] [2] [6] [9] [7] [4] [7] [2] [2] [3] [8] [3] [1]
[3] [5] [3] [0] [3] [0] [7] [0] [5] [4] [4] [5] [4] [1] [7] [9]
[5] [6] [6] [9] [7] [8] [5] [5] [4] [1] [8] [1] [9] [3] [0] [5]
[9] [5] [9] [2] [0] [3] [2] [1] [8] [5] [0] [5] [8] [2] [6] [3]
[7] [6] [8] [5] [0] [6] [0] [7] [9] [8] [7] [9] [7] [9] [3] [0]
[4] [0] [9] [5] [3] [4] [5] [9] [8] [6] [4] [4] [5] [9] [5] [3]
[2] [6] [1] [7] [5] [9] [0] [7] [9] [8] [6] [4] [1] [3] [6] [0]
[1] [6] [4] [0] [9] [4] [0] [0] [3] [8] [4] [6] [9] [8] [9] [6]
[9] [7] [2] [2] [8] [3] [8] [1] [7] [0] [3] [5] [8] [5] [0] [5]
[5] [9] [4] [7] [2] [0] [6] [1] [7] [4] [5] [3] [4] [1] [8] [5]
[8] [3] [0] [6] [1] [5] [4] [3] [0] [8] [9] [1] [5] [7] [6] [0]
[2] [8] [1] [7] [9] [0] [7] [3] [2] [5] [5] [5] [5] [2] [3] [1]
[9] [6] [8] [8] [2] [3] [1] [4] [1] [6] [4] [8] [9] [2] [8] [5]
[6] [2] [9] [9] [6] [7] [8] [6] [1] [7] [8] [0] [4] [8] [8] [8]
[8] [0] [4] [6] [1] [4] [8] [3] [9] [2] [3] [4] [3] [4] [1] [2]

B matrix:

[0] [4] [4] [3] [2] [9] [7] [9] [3] [3] [2] [2] [4] [2] [1] [4]
[9] [3] [6] [6] [9] [5] [0] [5] [2] [0] [5] [8] [5] [9] [3] [1]
[9] [6] [9] [8] [5] [2] [9] [8] [2] [5] [6] [1] [9] [7] [9] [4]
[2] [7] [9] [6] [9] [5] [0] [1] [2] [3] [7] [1] [8] [1] [5] [3]
[7] [6] [8] [9] [4] [7] [4] [1] [6] [9] [4] [0] [6] [5] [5] [6]
[4] [1] [8] [0] [6] [7] [9] [2] [5] [1] [5] [3] [1] [0] [0] [6]
[1] [9] [3] [6] [9] [3] [6] [2] [2] [2] [1] [1] [7] [1] [6] [6]
[5] [3] [7] [9] [4] [7] [3] [6] [2] [4] [3] [9] [4] [2] [5] [0]
[6] [2] [1] [4] [4] [0] [7] [4] [6] [2] [1] [4] [5] [4] [3] [3]
[6] [8] [0] [5] [0] [7] [5] [0] [3] [9] [2] [7] [4] [5] [1] [1]
[4] [3] [4] [2] [0] [5] [9] [8] [5] [1] [4] [7] [0] [5] [2] [7]
[0] [6] [9] [3] [4] [9] [2] [8] [9] [3] [3] [7] [4] [0] [0] [0]
[7] [5] [7] [4] [9] [7] [8] [1] [9] [5] [9] [8] [7] [5] [9] [0]
[4] [0] [8] [3] [7] [4] [5] [8] [4] [6] [1] [8] [4] [9] [6] [8]
[9] [9] [8] [3] [2] [4] [5] [3] [0] [6] [0] [2] [4] [8] [2] [4]
[0] [4] [9] [4] [4] [5] [1] [6] [9] [4] [7] [0] [9] [2] [6] [2]

Expected Result:

Resultant product **out_c** matrix [**C** = A*B]:

[287] [263] [425] [290] [320] [390] [332] [309] [333] [275] [270] [299] [336] [262] [256] [211]
[319] [305] [374] [322] [366] [371] [346] [298] [250] [257] [205] [353] [316] [297] [252] [248]
[265] [329] [353] [271] [274] [298] [284] [272] [285] [234] [212] [221] [333] [257] [239] [190]
[374] [358] [538] [396] [458] [443] [433] [346] [373] [284] [387] [329] [429] [308] [370] [300]
[345] [335] [402] [302] [328] [351] [384] [328] [295] [265] [259] [283] [374] [301] [275] [188]
[437] [375] [541] [395] [421] [494] [496] [468] [396] [341] [330] [494] [414] [394] [329] [285]
[394] [381] [510] [408] [393] [410] [457] [394] [336] [356] [284] [372] [428] [348] [365] [295]
[382] [324] [425] [331] [322] [397] [363] [287] [288] [281] [248] [356] [306] [297] [210] [248]
[427] [372] [506] [341] [351] [429] [387] [316] [380] [379] [296] [358] [384] [402] [308] [261]
[296] [322] [438] [342] [410] [399] [371] [340] [369] [260] [283] [292] [392] [285] [310] [263]
[346] [380] [413] [336] [359] [341] [320] [307] [272] [248] [261] [276] [387] [320] [277] [224]
[286] [315] [369] [258] [289] [389] [366] [289] [238] [263] [222] [326] [269] [285] [216] [264]
[312] [358] [393] [351] [352] [365] [270] [238] [271] [267] [252] [285] [342] [275] [262] [213]
[390] [441] [564] [387] [402] [495] [404] [396] [358] [339] [359] [362] [449] [346] [331] [229]
[437] [486] [629] [463] [462] [505] [510] [425] [364] [412] [371] [350] [507] [403] [420] [393]
[218] [288] [343] [275] [319] [306] [348] [295] [264] [207] [205] [230] [322] [193] [242] [234]

The simulation uses a **10 ns clock** (`clk`).

The ***reset* is cleared at the 5 ns mark**.

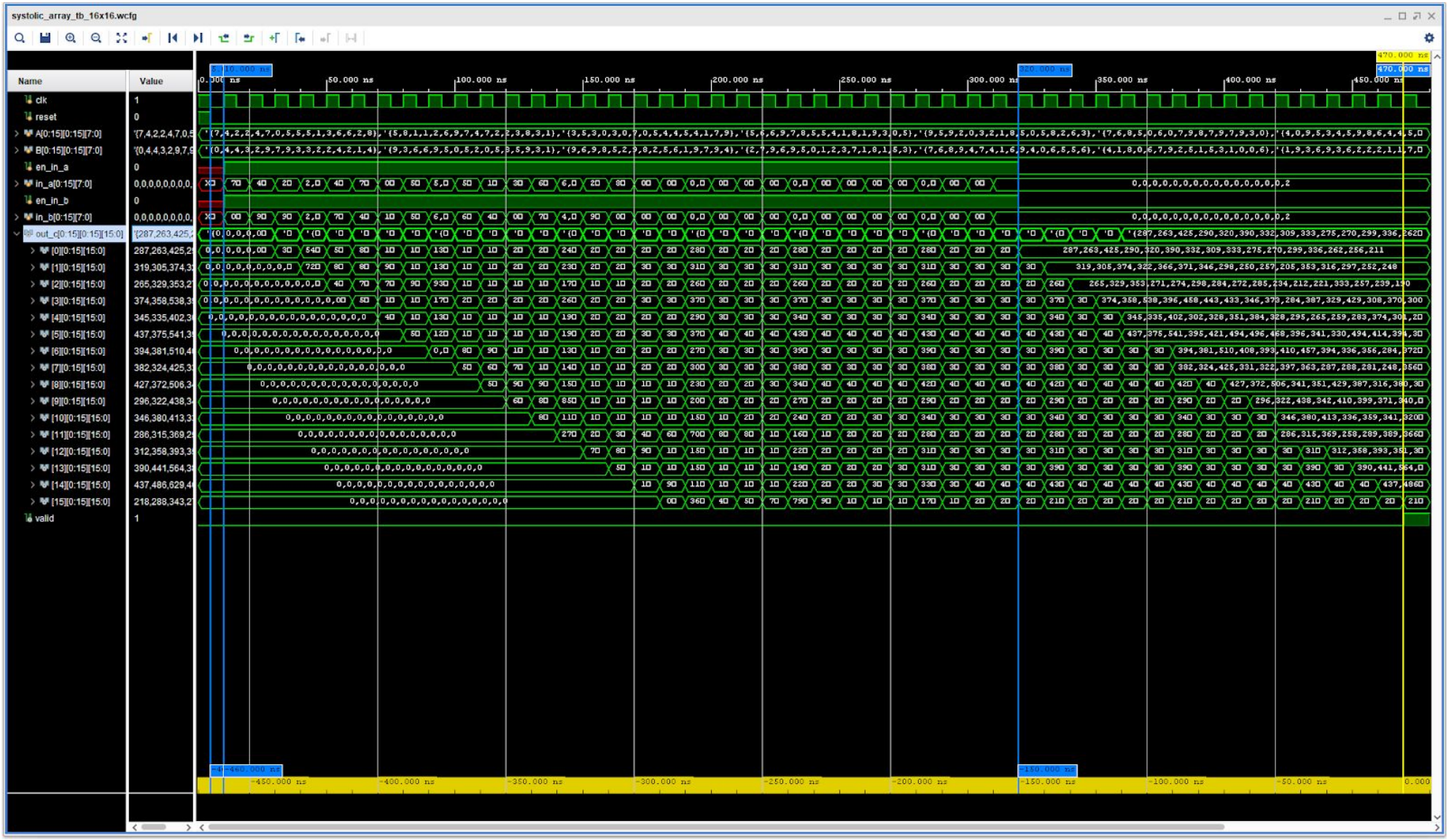
At the first positive clock after that (10 ns mark), the *en_in_a* and *en_in_b* lines are set to HIGH and the **first *in_a* and *in_b* data arrays are ingested by the systolic array**. At every following positive edge of the clock, new values are loaded from the A and B input matrices into the *in_a* and *in_b* data arrays.

***(2n-1) clock cycles* after the first ingestion** of first *in_a* and *in_b* data arrays (10 ns mark), the *n_in_a* and *en_in_b* lines are cleared, signifying the **ingestion of all of the input data is complete**. This occurs at the 3**20 ns mark**.

***(3n-2)* clock cycles after the first ingestion** (at the 470 ns mark), the ***valid* line goes HIGH**. This signifies that the **resulting product matrix is ready in the *out_c* bus**.

Since it is not possible to visualize the 256 input elements in the out_c bus, I used the testbench to output the values in the bus to the console. The data values match indicating a successful product computation. I paste the TCL console here after the waveform.

(The enlarged waveform is on the next page)


systolic_array_tb_16x16.wcfg
Name
Value
clk
reset
A[0:15][0:15][7:0]
B[0:15][0:15][7:0]
en_in_a
in_a[0:15][7:0]
en_in_b
in_b[0:15][7:0]
out_c[0:15][0:15][15:0]
[0][0:15][15:0]
[1][0:15][15:0]
[2][0:15][15:0]
[3][0:15][15:0]
[4][0:15][15:0]
[5][0:15][15:0]
[6][0:15][15:0]
[7][0:15][15:0]
[8][0:15][15:0]
[9][0:15][15:0]
[10][0:15][15:0]
[11][0:15][15:0]
[12][0:15][15:0]
[13][0:15][15:0]
[14][0:15][15:0]
[15][0:15][15:0]
valid
0.000 ns
50.000 ns
100.000 ns
150.000 ns
200.000 ns
250.000 ns
300.000 ns
350.000 ns
400.000 ns
450.000 ns
470.000 ns
-450.000 ns
-400.000 ns
-350.000 ns
-300.000 ns
-250.000 ns
-200.000 ns
-150.000 ns
-100.000 ns
-50.000 ns
0.000

TCL Console output (important entries highlighted)

```
-----------------------------------------
Matrix multiplication using systolic arrays:
-----------------------------------------

A matrix:
[  7] [  4] [  2] [  2] [  4] [  7] [  0] [  5] [  5] [  5] [  1] [  3] [  6] [  6] [  2] [  8]
[  5] [  8] [  1] [  1] [  2] [  6] [  9] [  7] [  4] [  7] [  2] [  2] [  3] [  8] [  3] [  1]
[  3] [  5] [  3] [  0] [  3] [  0] [  7] [  0] [  5] [  4] [  4] [  5] [  4] [  1] [  7] [  9]
[  5] [  6] [  6] [  9] [  7] [  8] [  5] [  5] [  4] [  1] [  8] [  1] [  9] [  3] [  0] [  5]
[  9] [  5] [  9] [  2] [  0] [  3] [  2] [  1] [  8] [  5] [  0] [  5] [  8] [  2] [  6] [  3]
[  7] [  6] [  8] [  5] [  0] [  6] [  0] [  7] [  9] [  8] [  7] [  9] [  7] [  9] [  3] [  0]
[  4] [  0] [  9] [  5] [  3] [  4] [  5] [  9] [  8] [  6] [  4] [  4] [  5] [  9] [  5] [  3]
[  2] [  6] [  1] [  7] [  5] [  9] [  0] [  7] [  9] [  8] [  6] [  4] [  1] [  3] [  6] [  0]
[  1] [  6] [  4] [  0] [  9] [  4] [  0] [  0] [  3] [  8] [  4] [  6] [  9] [  8] [  9] [  6]
[  9] [  7] [  2] [  2] [  8] [  3] [  8] [  1] [  7] [  0] [  3] [  5] [  8] [  5] [  0] [  5]
[  5] [  9] [  4] [  7] [  2] [  0] [  6] [  1] [  7] [  4] [  5] [  3] [  4] [  1] [  8] [  5]
[  8] [  3] [  0] [  6] [  1] [  5] [  4] [  3] [  0] [  8] [  9] [  1] [  5] [  7] [  6] [  0]
[  2] [  8] [  1] [  7] [  9] [  0] [  7] [  3] [  2] [  5] [  5] [  5] [  5] [  2] [  3] [  1]
[  9] [  6] [  8] [  8] [  2] [  3] [  1] [  4] [  1] [  6] [  4] [  8] [  9] [  2] [  8] [  5]
[  6] [  2] [  9] [  9] [  6] [  7] [  8] [  6] [  1] [  7] [  8] [  0] [  4] [  8] [  8] [  8]
[  8] [  0] [  4] [  6] [  1] [  4] [  8] [  3] [  9] [  2] [  3] [  4] [  3] [  4] [  1] [  2]



B matrix:
[  0] [  4] [  4] [  3] [  2] [  9] [  7] [  9] [  3] [  3] [  2] [  2] [  4] [  2] [  1] [  4]
[  9] [  3] [  6] [  6] [  9] [  5] [  0] [  5] [  2] [  0] [  5] [  8] [  5] [  9] [  3] [  1]
[  9] [  6] [  9] [  8] [  5] [  2] [  9] [  8] [  2] [  5] [  6] [  1] [  9] [  7] [  9] [  4]
[  2] [  7] [  9] [  6] [  9] [  5] [  0] [  1] [  2] [  3] [  7] [  1] [  8] [  1] [  5] [  3]
[  7] [  6] [  8] [  9] [  4] [  7] [  4] [  1] [  6] [  9] [  4] [  0] [  6] [  5] [  5] [  6]
[  4] [  1] [  8] [  0] [  6] [  7] [  9] [  2] [  5] [  1] [  5] [  3] [  1] [  0] [  0] [  6]
[  1] [  9] [  3] [  6] [  9] [  3] [  6] [  2] [  2] [  2] [  1] [  1] [  7] [  1] [  6] [  6]
[  5] [  3] [  7] [  9] [  4] [  7] [  3] [  6] [  2] [  4] [  3] [  9] [  4] [  2] [  5] [  0]
[  6] [  2] [  1] [  4] [  4] [  0] [  7] [  4] [  6] [  2] [  1] [  4] [  5] [  4] [  3] [  3]
[  6] [  8] [  0] [  5] [  0] [  7] [  5] [  0] [  3] [  9] [  2] [  7] [  4] [  5] [  1] [  1]
[  4] [  3] [  4] [  2] [  0] [  5] [  9] [  8] [  5] [  1] [  4] [  7] [  0] [  5] [  2] [  7]
[  0] [  6] [  9] [  3] [  4] [  9] [  2] [  8] [  9] [  3] [  3] [  7] [  4] [  0] [  0] [  0]
[  7] [  5] [  7] [  4] [  9] [  7] [  8] [  1] [  9] [  5] [  9] [  8] [  7] [  5] [  9] [  0]
[  4] [  0] [  8] [  3] [  7] [  4] [  5] [  8] [  4] [  6] [  1] [  8] [  4] [  9] [  6] [  8]
[  9] [  9] [  8] [  3] [  2] [  4] [  5] [  3] [  0] [  6] [  0] [  2] [  4] [  8] [  2] [  4]
[  0] [  4] [  9] [  4] [  4] [  5] [  1] [  6] [  9] [  4] [  7] [  0] [  9] [  2] [  6] [  2]


Time: CLK RESET | EN_IN_A EN_IN_B | VALID
0:    1    1 | x    x | 0
5:    0    0 | x    x | 0

A input        0: [  7] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]
B input        0: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]

10:    1    0 | 1    1 | 0
15:    0    0 | 1    1 | 0

A input        1: [  4] [  5] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]
B input        1: [  9] [  4] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]

20:    1    0 | 1    1 | 0
```

```
25:    0    0 | 1    1 | 0

A input       2: [  2] [  8] [  3] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]
B input       2: [  9] [  3] [  4] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]

30:    1    0 | 1    1 | 0
35:    0    0 | 1    1 | 0

A input       3: [  2] [  1] [  5] [  5] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]
B input       3: [  2] [  6] [  6] [  3] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]

40:    1    0 | 1    1 | 0
45:    0    0 | 1    1 | 0

A input       4: [  4] [  1] [  3] [  6] [  9] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]
B input       4: [  7] [  7] [  9] [  6] [  2] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]

50:    1    0 | 1    1 | 0
55:    0    0 | 1    1 | 0

A input       5: [  7] [  2] [  0] [  6] [  5] [  7] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]
B input       5: [  4] [  6] [  9] [  8] [  9] [  9] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]

60:    1    0 | 1    1 | 0
65:    0    0 | 1    1 | 0

A input       6: [  0] [  6] [  3] [  9] [  9] [  6] [  4] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]
B input       6: [  1] [  1] [  8] [  6] [  5] [  5] [  7] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]

70:    1    0 | 1    1 | 0
75:    0    0 | 1    1 | 0

A input       7: [  5] [  9] [  0] [  7] [  2] [  8] [  0] [  2] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]
B input       7: [  5] [  9] [  8] [  9] [  9] [  2] [  0] [  9] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]

80:    1    0 | 1    1 | 0
85:    0    0 | 1    1 | 0

A input       8: [  5] [  7] [  7] [  8] [  0] [  5] [  9] [  6] [  1] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]
B input       8: [  6] [  3] [  3] [  0] [  4] [  5] [  9] [  5] [  3] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
0]

90:    1    0 | 1    1 | 0
95:    0    0 | 1    1 | 0

A input       9: [  5] [  4] [  0] [  5] [  3] [  0] [  5] [  1] [  6] [  9] [  0] [  0] [  0] [  0] [  0] [  0] [
0]
B input       9: [  6] [  2] [  7] [  6] [  6] [  7] [  0] [  8] [  2] [  3] [  0] [  0] [  0] [  0] [  0] [  0] [
0]

100:    1    0 | 1    1 | 0
```

```
105:    0    0 | 1    1 | 0

A input       10: [  1] [  7] [  5] [  5] [  2] [  6] [  3] [  7] [  4] [  7] [  5] [  0] [  0] [  0] [  0] [
0]
B input       10: [  4] [  8] [  1] [  9] [  9] [  7] [  4] [  1] [  2] [  0] [  2] [  0] [  0] [  0] [  0] [
0]

110:    1    0 | 1    1 | 0
115:    0    0 | 1    1 | 0

A input       11: [  3] [  2] [  4] [  4] [  1] [  0] [  4] [  5] [  0] [  2] [  9] [  8] [  0] [  0] [  0] [
0]
B input       11: [  0] [  3] [  0] [  4] [  4] [  3] [  9] [  1] [  2] [  5] [  5] [  2] [  0] [  0] [  0] [
0]

120:    1    0 | 1    1 | 0
125:    0    0 | 1    1 | 0

A input       12: [  6] [  2] [  4] [  1] [  8] [  7] [  5] [  9] [  9] [  2] [  4] [  3] [  2] [  0] [  0] [
0]
B input       12: [  7] [  6] [  4] [  5] [  4] [  7] [  6] [  2] [  6] [  3] [  6] [  8] [  4] [  0] [  0] [
0]

130:    1    0 | 1    1 | 0
135:    0    0 | 1    1 | 0

A input       13: [  6] [  3] [  5] [  8] [  5] [  9] [  9] [  0] [  4] [  8] [  7] [  0] [  8] [  9] [  0] [
0]
B input       13: [  4] [  5] [  9] [  2] [  0] [  0] [  3] [  2] [  5] [  9] [  7] [  1] [  5] [  2] [  0] [
0]

140:    1    0 | 1    1 | 0
145:    0    0 | 1    1 | 0

A input       14: [  2] [  8] [  4] [  1] [  0] [  8] [  8] [  7] [  0] [  3] [  2] [  6] [  1] [  6] [  6] [
0]
B input       14: [  9] [  0] [  7] [  3] [  0] [  7] [  7] [  6] [  2] [  1] [  4] [  1] [  9] [  9] [  1] [
0]

150:    1    0 | 1    1 | 0
155:    0    0 | 1    1 | 0

A input       15: [  8] [  3] [  1] [  9] [  5] [  7] [  6] [  9] [  0] [  8] [  0] [  1] [  7] [  8] [  2] [
8]
B input       15: [  0] [  9] [  8] [  4] [  4] [  5] [  5] [  4] [  2] [  2] [  5] [  0] [  8] [  7] [  3] [
4]

160:    1    0 | 1    1 | 0
165:    0    0 | 1    1 | 0

A input       16: [  0] [  1] [  7] [  3] [  8] [  9] [  4] [  8] [  3] [  1] [  6] [  5] [  9] [  8] [  9] [
0]
B input       16: [  0] [  4] [  8] [  3] [  9] [  9] [  9] [  0] [  6] [  4] [  1] [  3] [  6] [  1] [  9] [
1]

170:    1    0 | 1    1 | 0
175:    0    0 | 1    1 | 0

A input       17: [  0] [  0] [  9] [  0] [  2] [  7] [  4] [  6] [  8] [  7] [  1] [  4] [  0] [  2] [  9] [
4]
B input       17: [  0] [  0] [  9] [  3] [  7] [  7] [  2] [  8] [  3] [  2] [  3] [  1] [  1] [  5] [  5] [
4]

180:    1    0 | 1    1 | 0
```

```
185:    0    0 | 1    1 | 0

A input       18: [  0] [  0] [  0] [  5] [  6] [  9] [  5] [  4] [  4] [  0] [  7] [  3] [  7] [  3] [  6] [
6]
B input       18: [  0] [  0] [  0] [  4] [  2] [  4] [  8] [  8] [  5] [  9] [  1] [  9] [  7] [  0] [  5] [
3]

190:    1    0 | 1    1 | 0
195:    0    0 | 1    1 | 0

A input       19: [  0] [  0] [  0] [  0] [  3] [  3] [  9] [  1] [  6] [  3] [  4] [  0] [  3] [  1] [  7] [
1]
B input       19: [  0] [  0] [  0] [  0] [  4] [  4] [  5] [  1] [  9] [  1] [  2] [  4] [  4] [  1] [  0] [
6]

200:    1    0 | 1    1 | 0
205:    0    0 | 1    1 | 0

A input       20: [  0] [  0] [  0] [  0] [  0] [  0] [  5] [  3] [  9] [  5] [  5] [  8] [  2] [  4] [  8] [
4]
B input       20: [  0] [  0] [  0] [  0] [  0] [  5] [  5] [  8] [  9] [  3] [  4] [  7] [  5] [  2] [  6] [
6]

210:    1    0 | 1    1 | 0
215:    0    0 | 1    1 | 0

A input       21: [  0] [  0] [  0] [  0] [  0] [  0] [  3] [  6] [  8] [  8] [  3] [  9] [  5] [  1] [  6] [
8]
B input       21: [  0] [  0] [  0] [  0] [  0] [  0] [  1] [  3] [  4] [  5] [  3] [  7] [  4] [  4] [  5] [
6]

220:    1    0 | 1    1 | 0
225:    0    0 | 1    1 | 0

A input       22: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  9] [  5] [  4] [  1] [  5] [  6] [  1] [
3]
B input       22: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  6] [  0] [  6] [  9] [  7] [  0] [  5] [  3] [
0]

230:    1    0 | 1    1 | 0
235:    0    0 | 1    1 | 0

A input       23: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  6] [  0] [  1] [  5] [  5] [  4] [  7] [
9]
B input       23: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  9] [  6] [  1] [  8] [  4] [  5] [  1] [
3]

240:    1    0 | 1    1 | 0
245:    0    0 | 1    1 | 0

A input       24: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  5] [  8] [  7] [  5] [  8] [  8] [
2]
B input       24: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  4] [  0] [  8] [  7] [  0] [  2] [
1]

250:    1    0 | 1    1 | 0
255:    0    0 | 1    1 | 0

A input       25: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  5] [  6] [  2] [  9] [  0] [
3]
B input       25: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  7] [  2] [  4] [  5] [  0] [
7]

260:    1    0 | 1    1 | 0
```

```
265:    0    0 | 1    1 | 0

A input       26: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  3] [  2] [  4] [
4]
B input       26: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  4] [  9] [  9] [
0]

270:    1    0 | 1    1 | 0
275:    0    0 | 1    1 | 0

A input       27: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  1] [  8] [  8] [
3]
B input       27: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  9] [  8] [  6] [
0]

280:    1    0 | 1    1 | 0
285:    0    0 | 1    1 | 0

A input       28: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  5] [  8] [
4]
B input       28: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  2] [  2] [
8]

290:    1    0 | 1    1 | 0
295:    0    0 | 1    1 | 0

A input       29: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  8] [
1]
B input       29: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  6] [
4]

300:    1    0 | 1    1 | 0
305:    0    0 | 1    1 | 0

A input       30: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
2]
B input       30: [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [  0] [
2]

310:    1    0 | 1    1 | 0
315:    0    0 | 1    1 | 0
320:    1    0 | 0    0 | 0
325:    0    0 | 0    0 | 0
330:    1    0 | 0    0 | 0
335:    0    0 | 0    0 | 0
340:    1    0 | 0    0 | 0
345:    0    0 | 0    0 | 0
350:    1    0 | 0    0 | 0
355:    0    0 | 0    0 | 0
360:    1    0 | 0    0 | 0
365:    0    0 | 0    0 | 0
370:    1    0 | 0    0 | 0
375:    0    0 | 0    0 | 0
380:    1    0 | 0    0 | 0
385:    0    0 | 0    0 | 0
390:    1    0 | 0    0 | 0
395:    0    0 | 0    0 | 0
400:    1    0 | 0    0 | 0
405:    0    0 | 0    0 | 0
410:    1    0 | 0    0 | 0
415:    0    0 | 0    0 | 0
420:    1    0 | 0    0 | 0
425:    0    0 | 0    0 | 0
430:    1    0 | 0    0 | 0
```

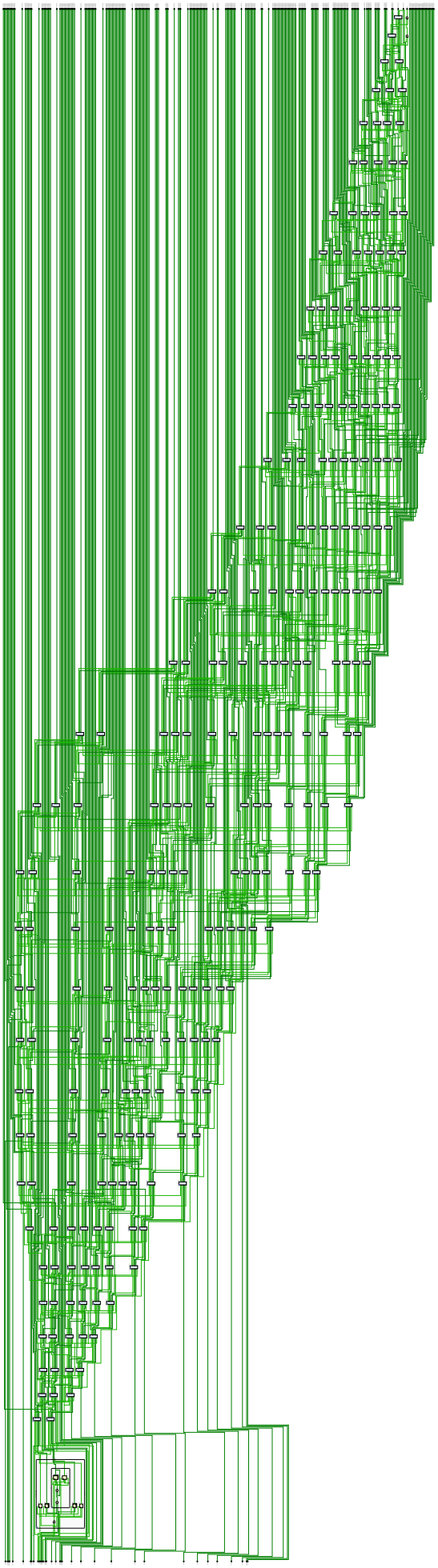
```
435:    0    0 | 0    0 | 0
440:    1    0 | 0    0 | 0
445:    0    0 | 0    0 | 0
450:    1    0 | 0    0 | 0
455:    0    0 | 0    0 | 0
460:    1    0 | 0    0 | 0
465:    0    0 | 0    0 | 0

Resultant product matrix [C = A*B]:
[  287] [  263] [  425] [  290] [  320] [  390] [  332] [  309] [  333] [  275] [  270] [  299] [  336] [  262]
[  256] [  211]
[  319] [  305] [  374] [  322] [  366] [  371] [  346] [  298] [  250] [  257] [  205] [  353] [  316] [  297]
[  252] [  248]
[  265] [  329] [  353] [  271] [  274] [  298] [  284] [  272] [  285] [  234] [  212] [  221] [  333] [  257]
[  239] [  190]
[  374] [  358] [  538] [  396] [  458] [  443] [  433] [  346] [  373] [  284] [  387] [  329] [  429] [  308]
[  370] [  300]
[  345] [  335] [  402] [  302] [  328] [  351] [  384] [  328] [  295] [  265] [  259] [  283] [  374] [  301]
[  275] [  188]
[  437] [  375] [  541] [  395] [  421] [  494] [  496] [  468] [  396] [  341] [  330] [  494] [  414] [  394]
[  329] [  285]
[  394] [  381] [  510] [  408] [  393] [  410] [  457] [  394] [  336] [  356] [  284] [  372] [  428] [  348]
[  365] [  295]
[  382] [  324] [  425] [  331] [  322] [  397] [  363] [  287] [  288] [  281] [  248] [  356] [  306] [  297]
[  210] [  248]
[  427] [  372] [  506] [  341] [  351] [  429] [  387] [  316] [  380] [  379] [  296] [  358] [  384] [  402]
[  308] [  261]
[  296] [  322] [  438] [  342] [  410] [  399] [  371] [  340] [  369] [  260] [  283] [  292] [  392] [  285]
[  310] [  263]
[  346] [  380] [  413] [  336] [  359] [  341] [  320] [  307] [  272] [  248] [  261] [  276] [  387] [  320]
[  277] [  224]
[  286] [  315] [  369] [  258] [  289] [  389] [  366] [  289] [  238] [  263] [  222] [  326] [  269] [  285]
[  216] [  264]
[  312] [  358] [  393] [  351] [  352] [  365] [  270] [  238] [  271] [  267] [  252] [  285] [  342] [  275]
[  262] [  213]
[  390] [  441] [  564] [  387] [  402] [  495] [  404] [  396] [  358] [  339] [  359] [  362] [  449] [  346]
[  331] [  229]
[  437] [  486] [  629] [  463] [  462] [  505] [  510] [  425] [  364] [  412] [  371] [  350] [  507] [  403]
[  420] [  393]
[  218] [  288] [  343] [  275] [  319] [  306] [  348] [  295] [  264] [  207] [  205] [  230] [  322] [  193]
[  242] [  234]

470:    1    0 | 0    0 | 1
475:    0    0 | 0    0 | 1
$finish called at time : 480 ns : File "C:/Users/tyros/codebase/EE599/HW2/code/systolic array/systolic
array.srcs/sim_1/new/systolic_array_tb.sv" Line 170
```

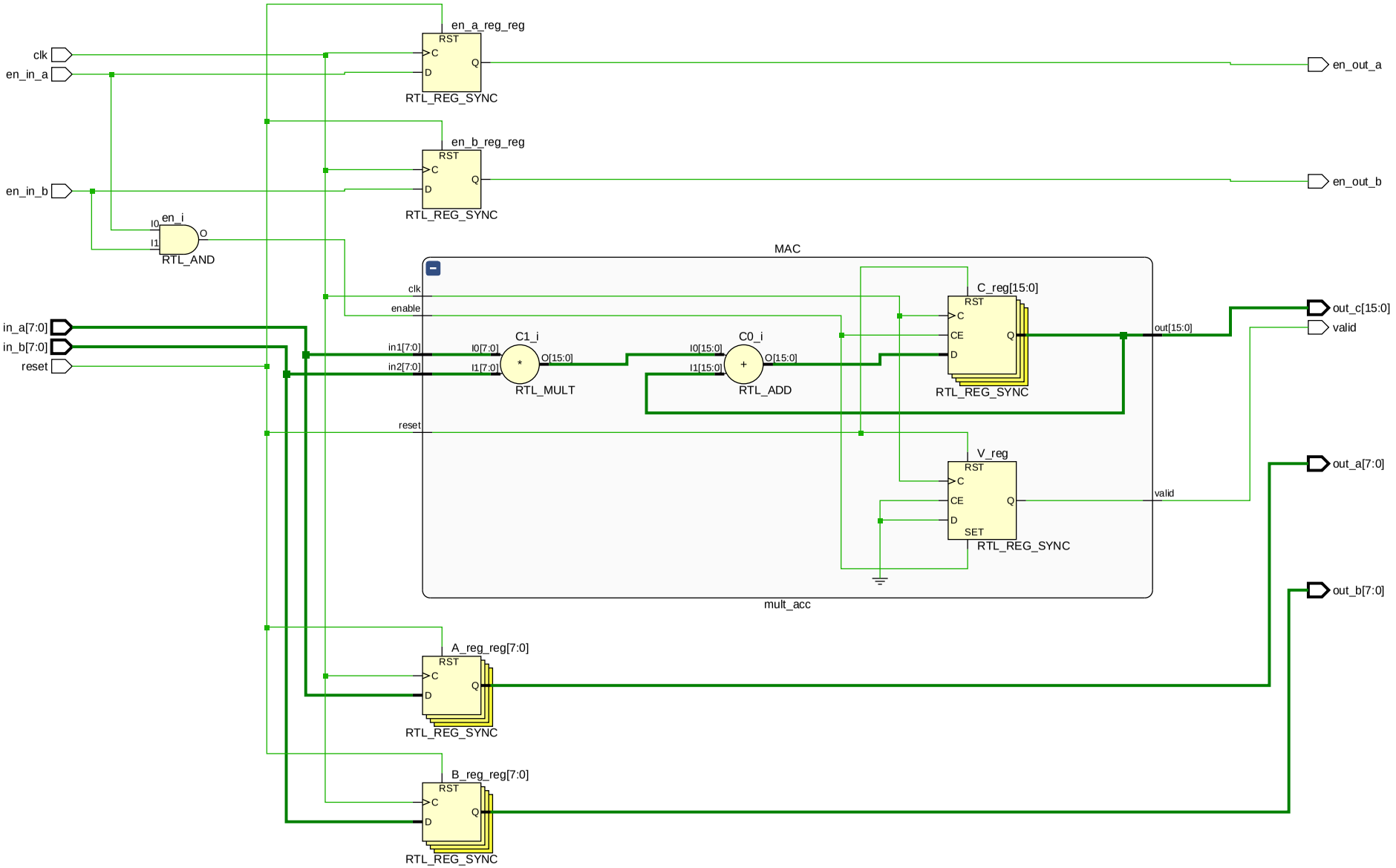
Elaborated Design (16x16 matrices - full systolic array)

(Inserted in the next page - for brevity only one systolic cell PE is expanded. However, the Multiply-Accumulate module within the PE is expanded)



Elaborated Design (16x16 matrices - single systolic cell PE + internal MAC)

(Inserted in the next page)


clk
en_in_a
en_a_reg_reg
RST
C
D
Q
RTL_REG_SYNC
en_out_a
en_b_reg_reg
en_in_b
en_out_b
en_i
I0
I1
O
RTL_AND
MAC
clk
enable
in_a[7:0]
in_b[7:0]
reset
in1[7:0]
in2[7:0]
C1_i
I0[7:0]
I1[7:0]
O[15:0]
*
RTL_MULT
C0_i
I0[15:0]
I1[15:0]
+
RTL_ADD
C_reg[15:0]
CE
out[15:0]
out_c[15:0]
valid
V_reg
SET
out_a[7:0]
out_b[7:0]
mult_acc
A_reg_reg[7:0]
B_reg_reg[7:0]

Synthesized Schematic (16x16 matrices)

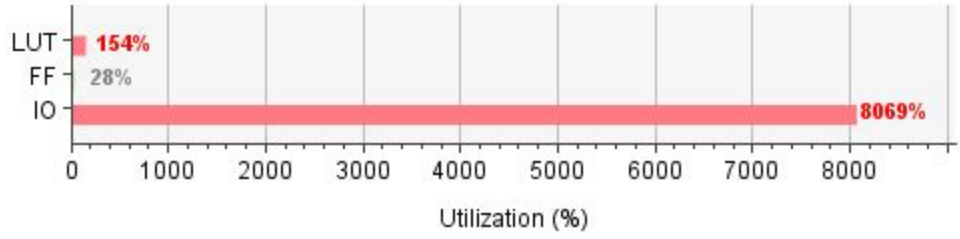
(Schematic is 15 pages long - inserted in the **Appendix B**)

Synthesis Reports (16x16 matrices)

Resource utilization report (16x16 matrices)

Summary

Resource	Utilization	Available	Utilization %
LUT	22146	14400	153.79
FF	7967	28800	27.66
IO	4357	54	8068.52



Timing estimation report (16x16 matrices)

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3.728 ns	Worst Hold Slack (WHS):	0.115 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	11789	Total Number of Endpoints:	11789	Total Number of Endpoints:	7968

All user specified timing constraints are met.

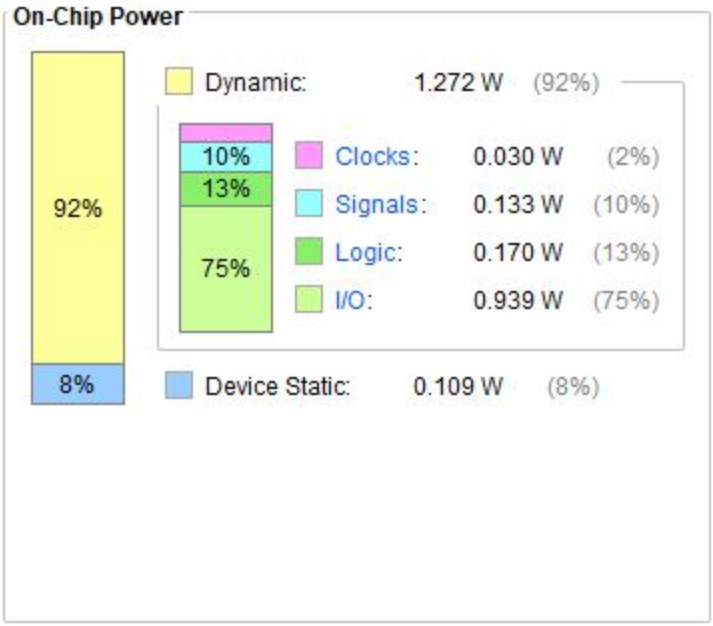
Power estimation report (16x16 matrices)

Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**1.381 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**40.9°C**
Thermal Margin:	59.1°C (4.9 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



Elaborated Design (32x32 matrices - full systolic array)

(Inserted in the next page - for brevity only one systolic cell PE is expanded. However, the Multiply-Accumulate module within the PE is expanded)

Elaborated Design (32x32 matrices - single systolic cell PE + internal MAC)

(Not inserted for brevity - same at 16x16 design)

Synthesized Schematic (32x32 matrices)

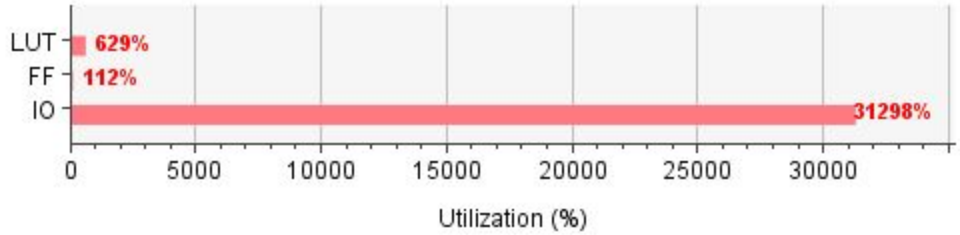
(Schematic is 58 pages long - inserted in **Appendix C**)

Synthesis Reports (32x32 matrices)

Resource utilization report (32x32 matrices)

Summary

Resource	Utilization	Available	Utilization %
LUT	90577	14400	629.01
FF	32319	28800	112.22
IO	16901	54	31298.15



Timing estimation report (32x32 matrices)

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3.813 ns	Worst Hold Slack (WHS):	0.115 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	48173	Total Number of Endpoints:	48173	Total Number of Endpoints:	32320

All user specified timing constraints are met.

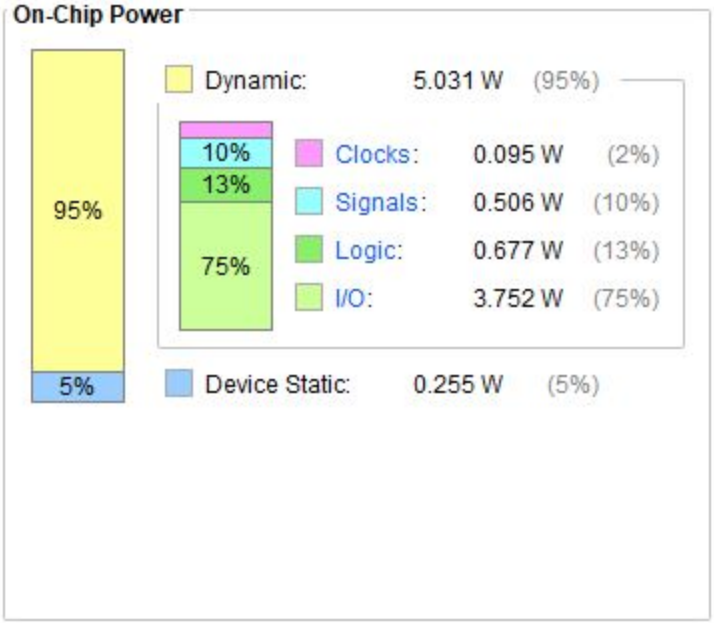
Power estimation report (32x32 matrices)

Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**5.286 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**86.0°C**
Thermal Margin:	14.0°C (1.2 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

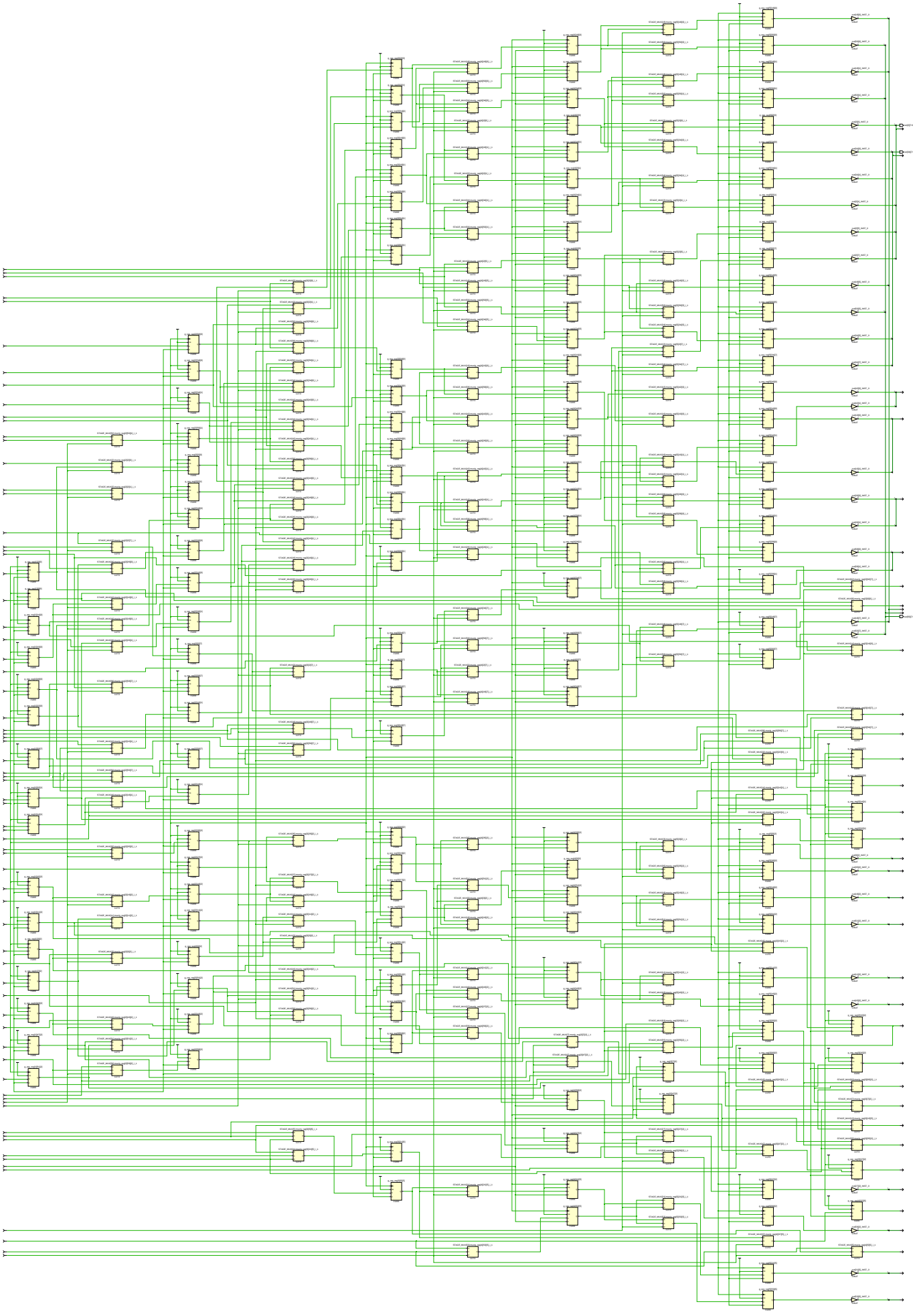
Launch Power Constraint Advisor to find and fix invalid switching activity

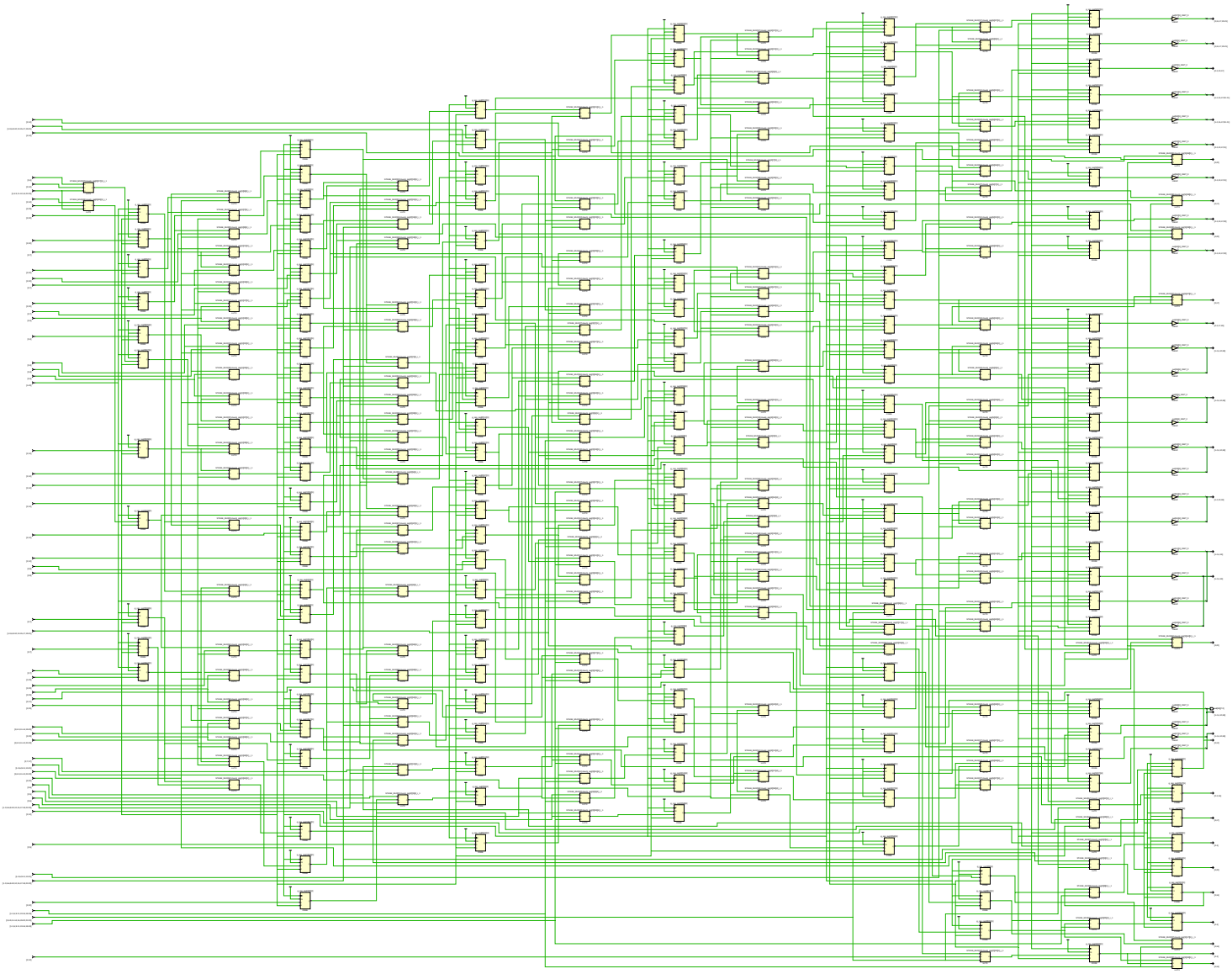


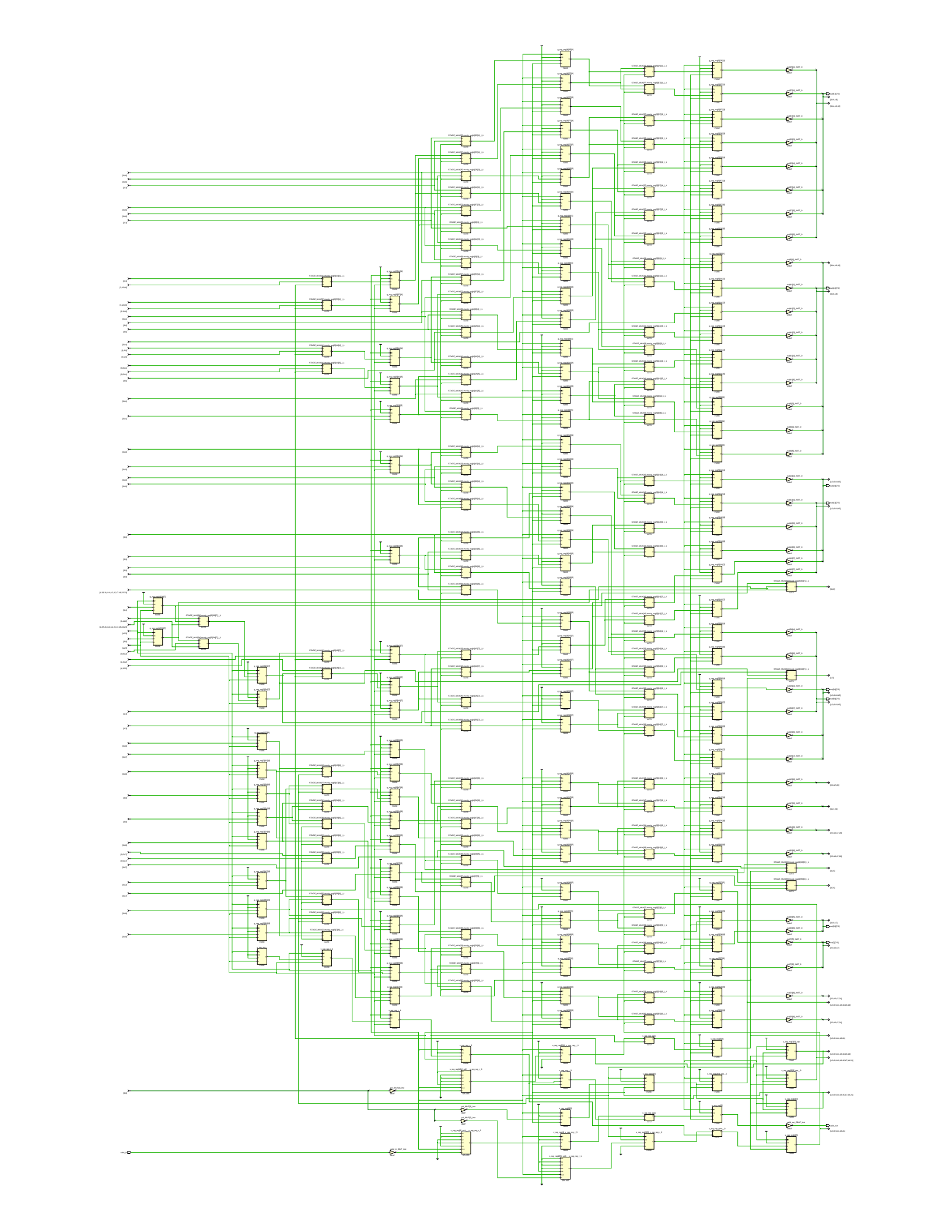
Appendix A: Barrel Shifter - 64 element sequence

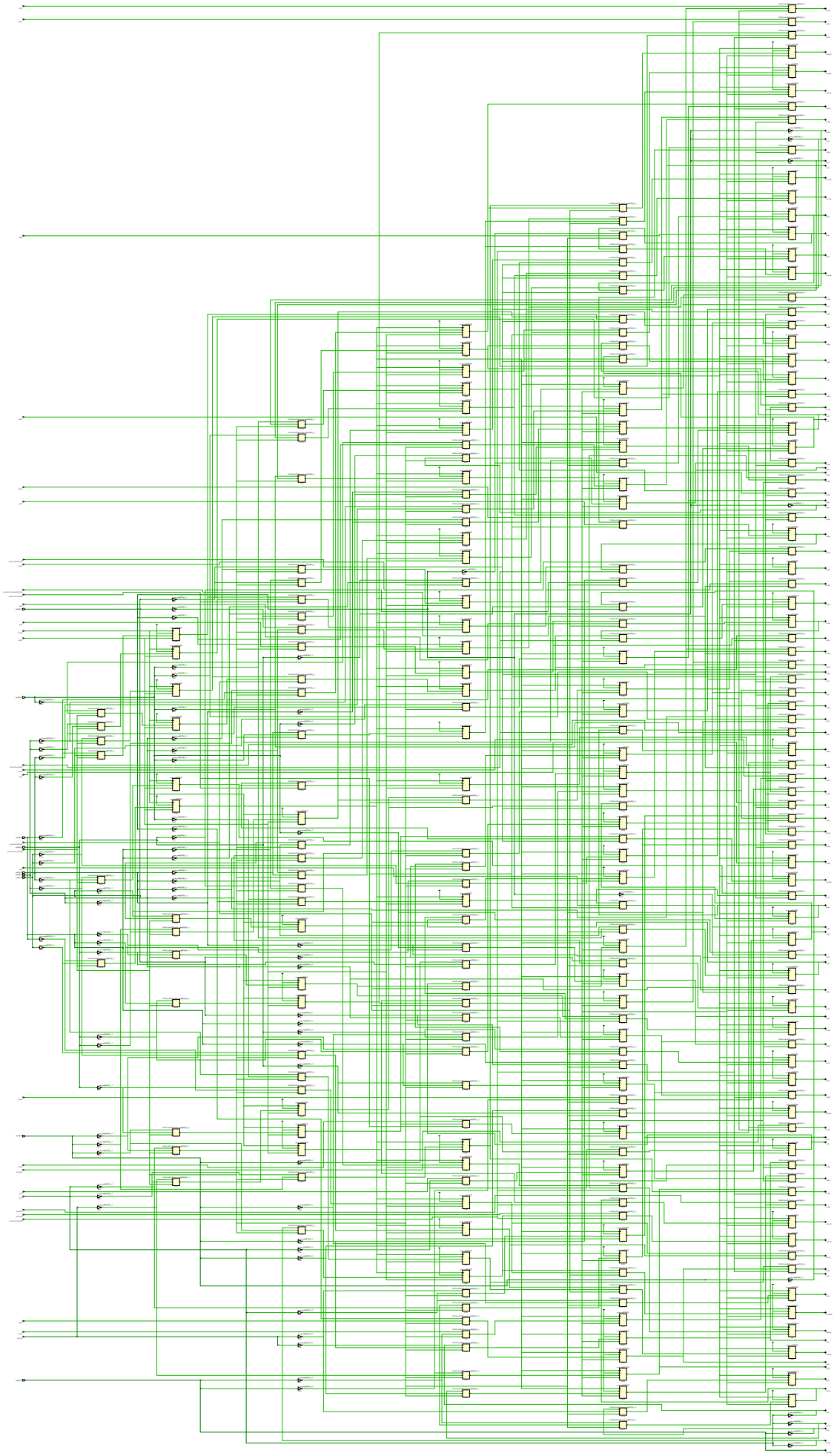
Synthesized Schematic

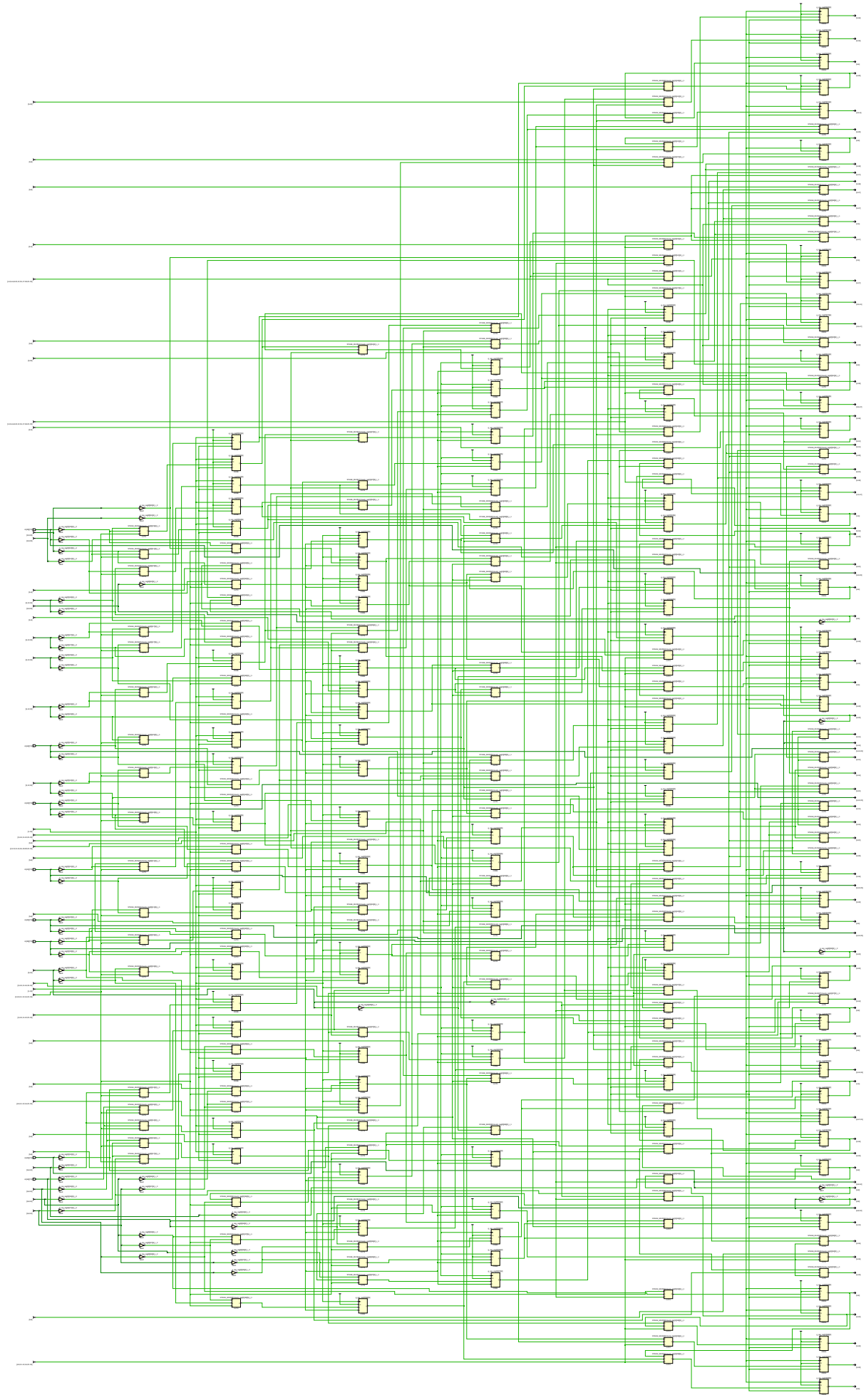
(Inserted in the next page)

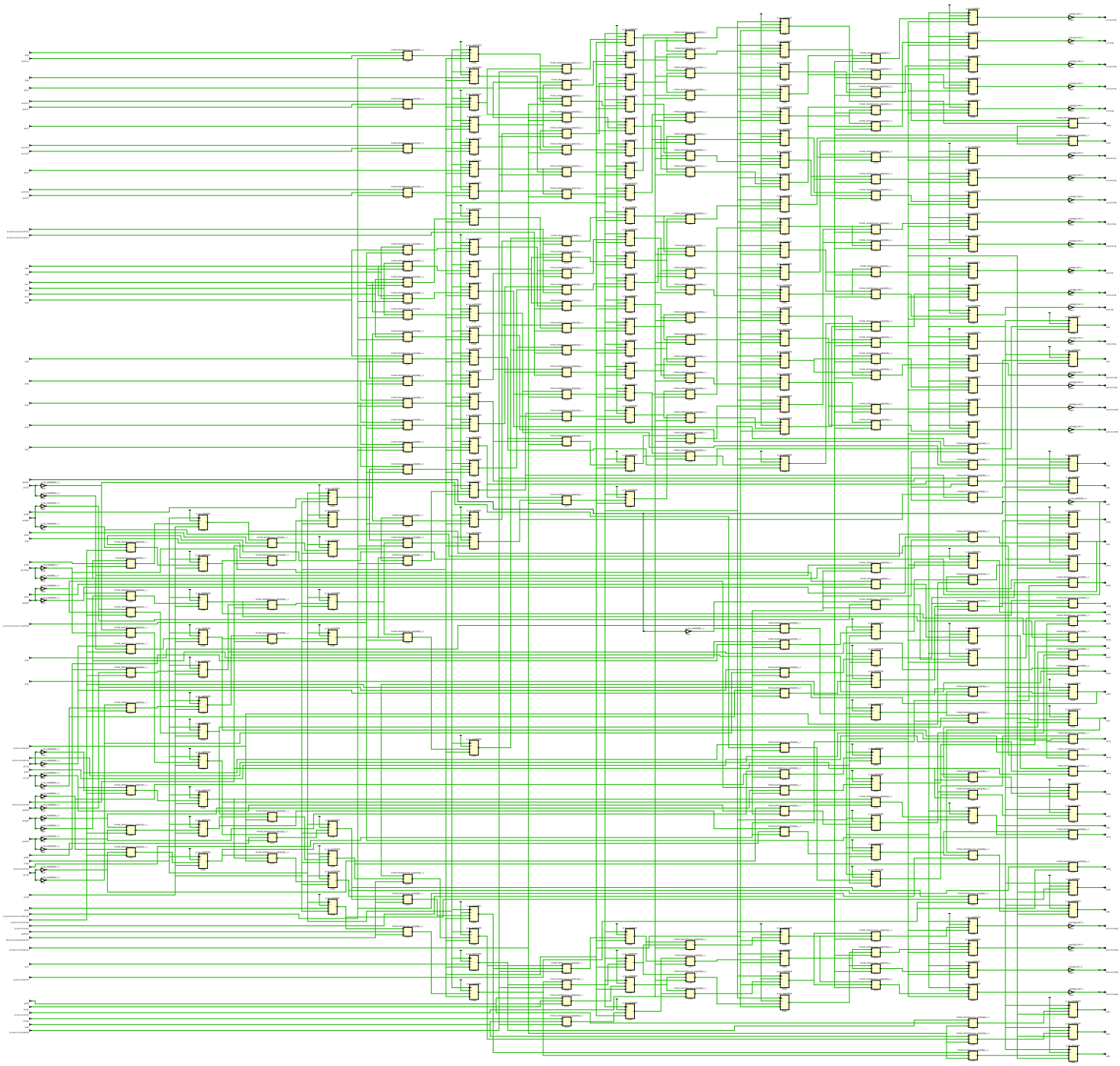


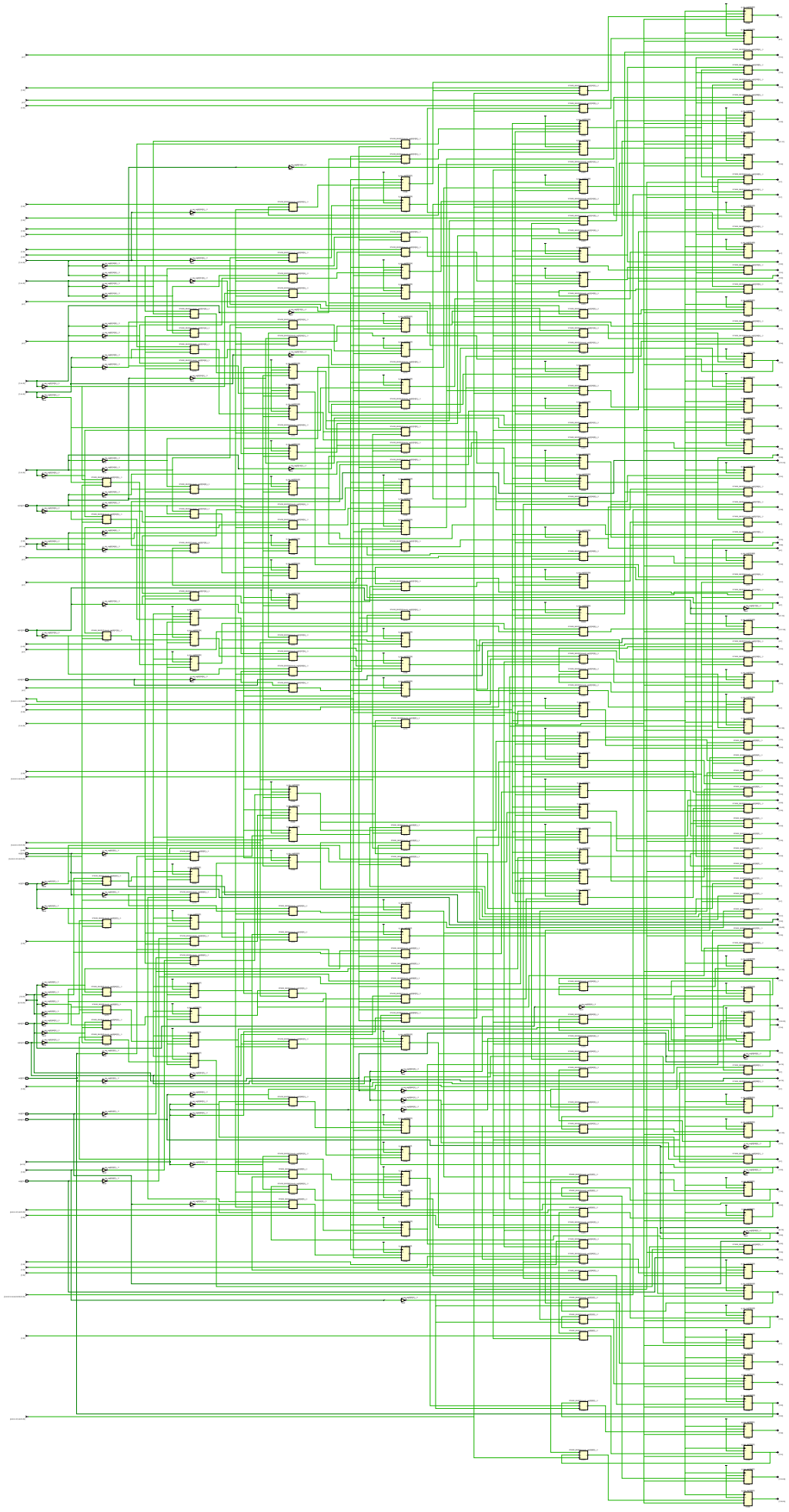


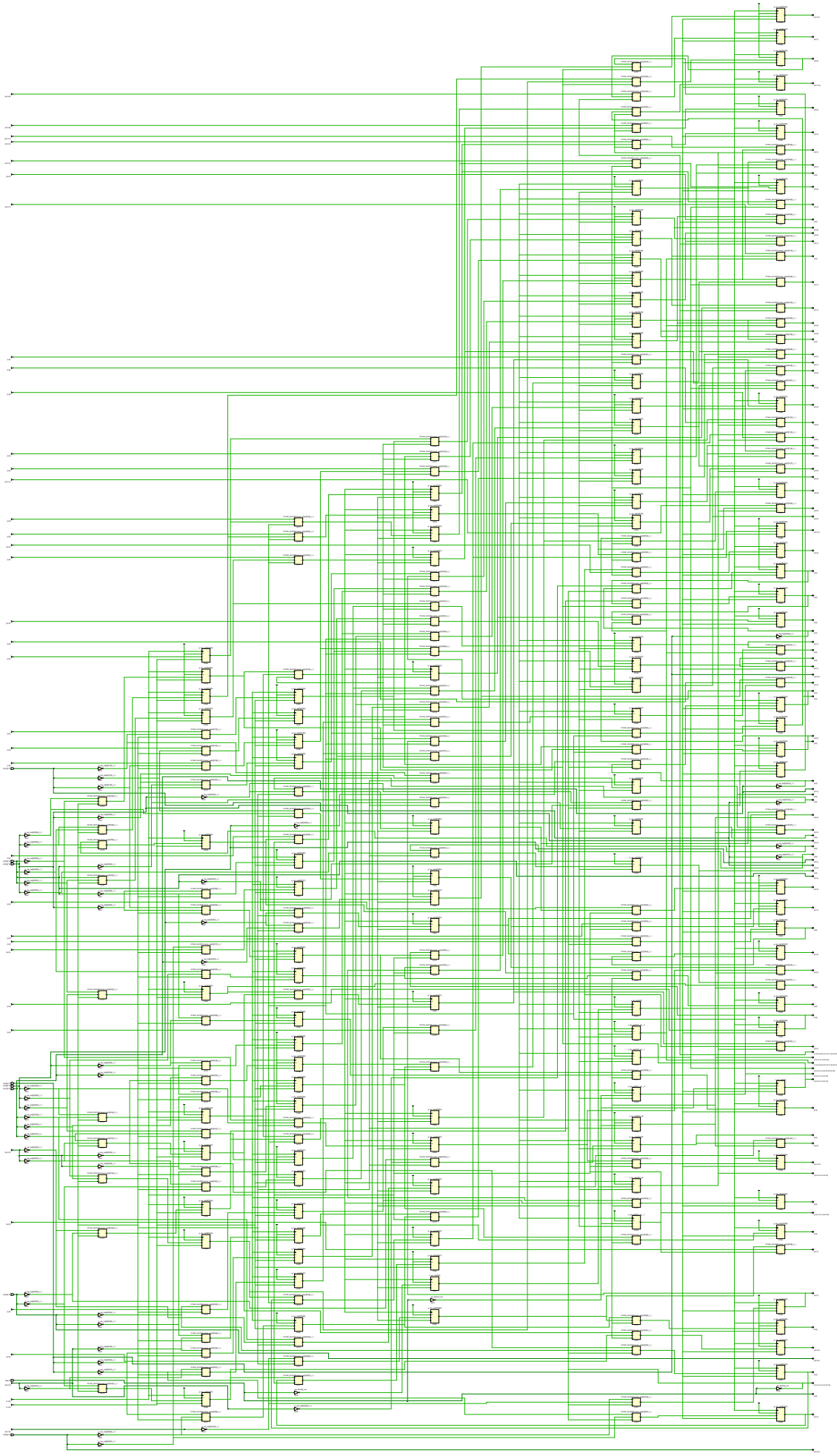


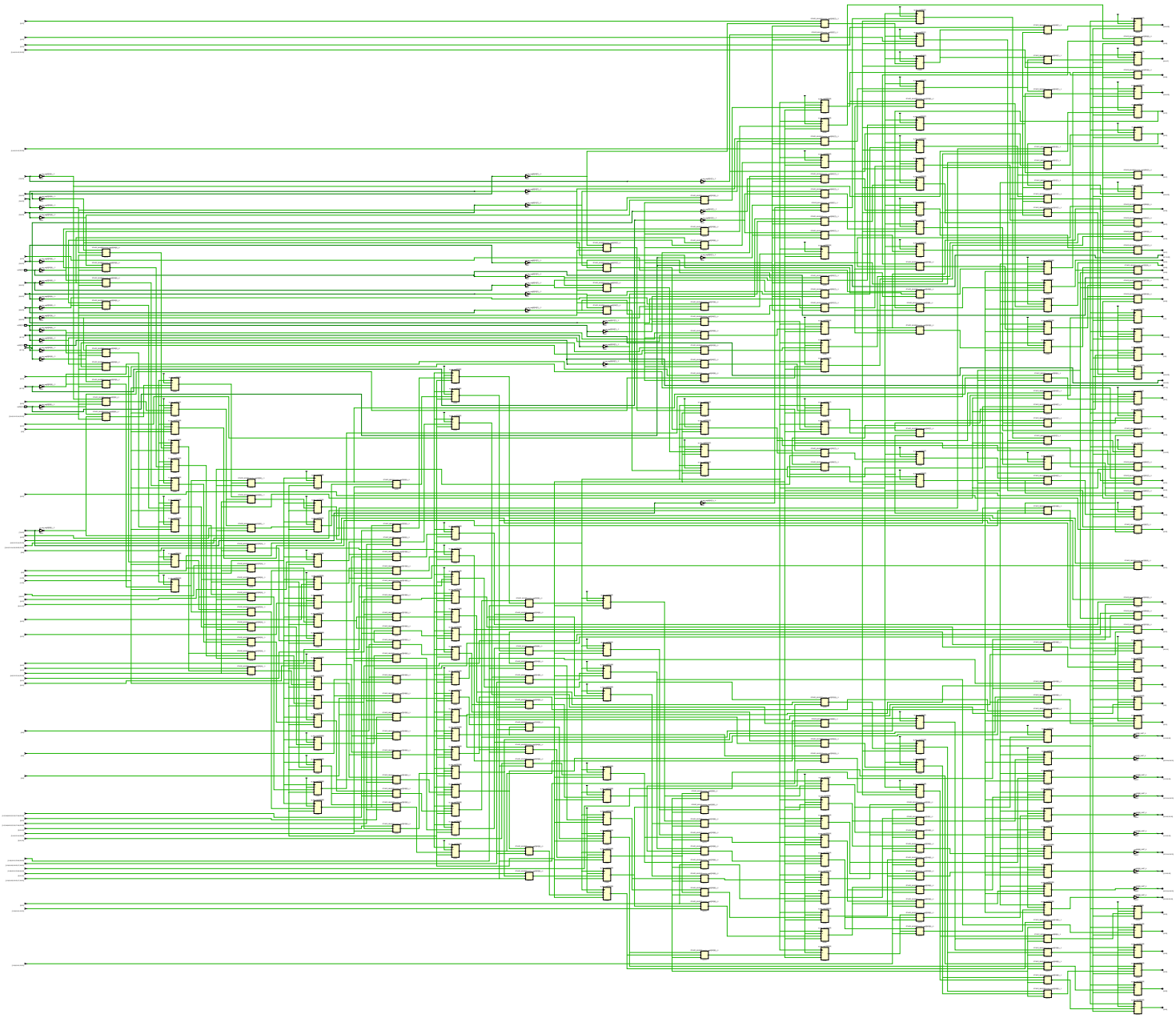


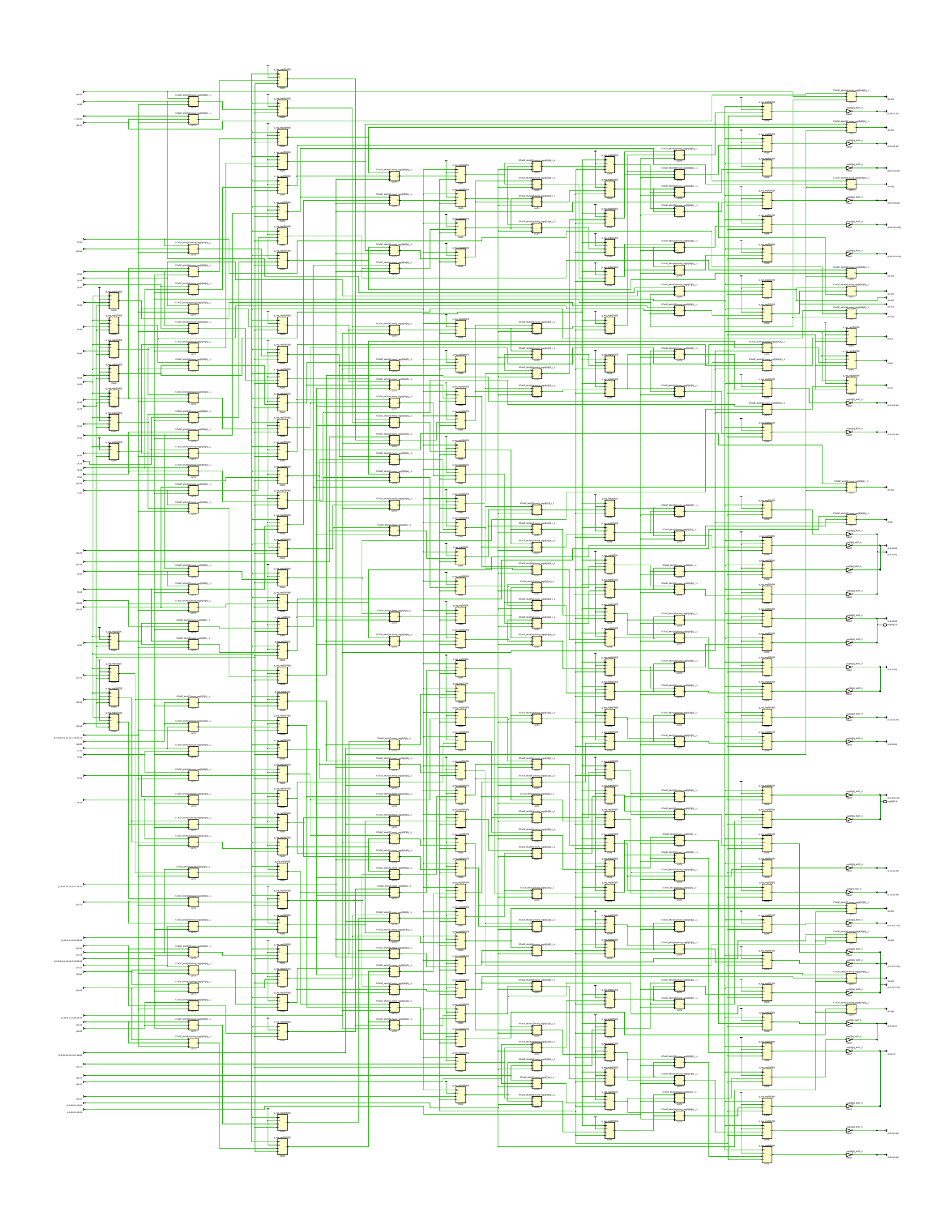


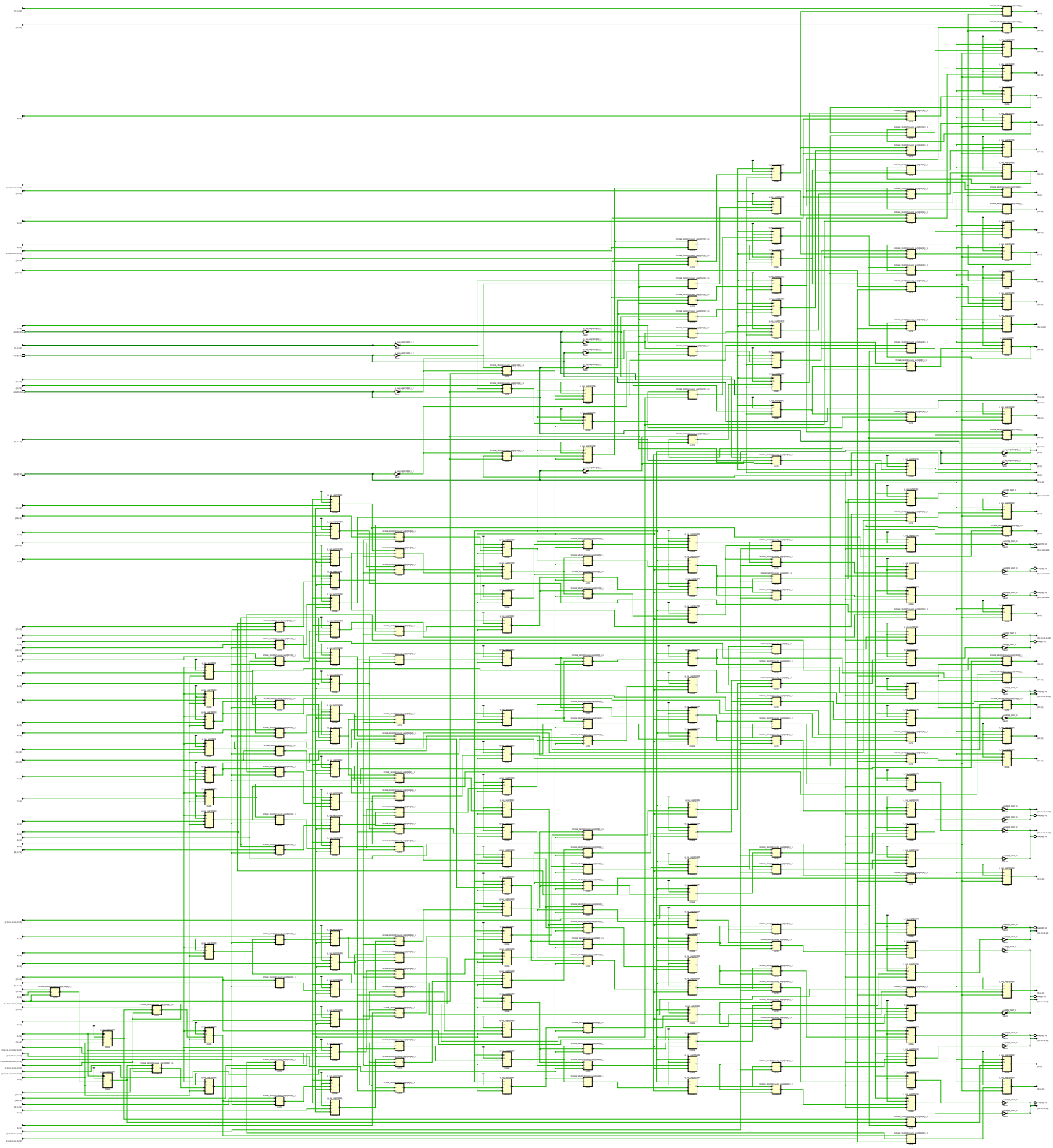


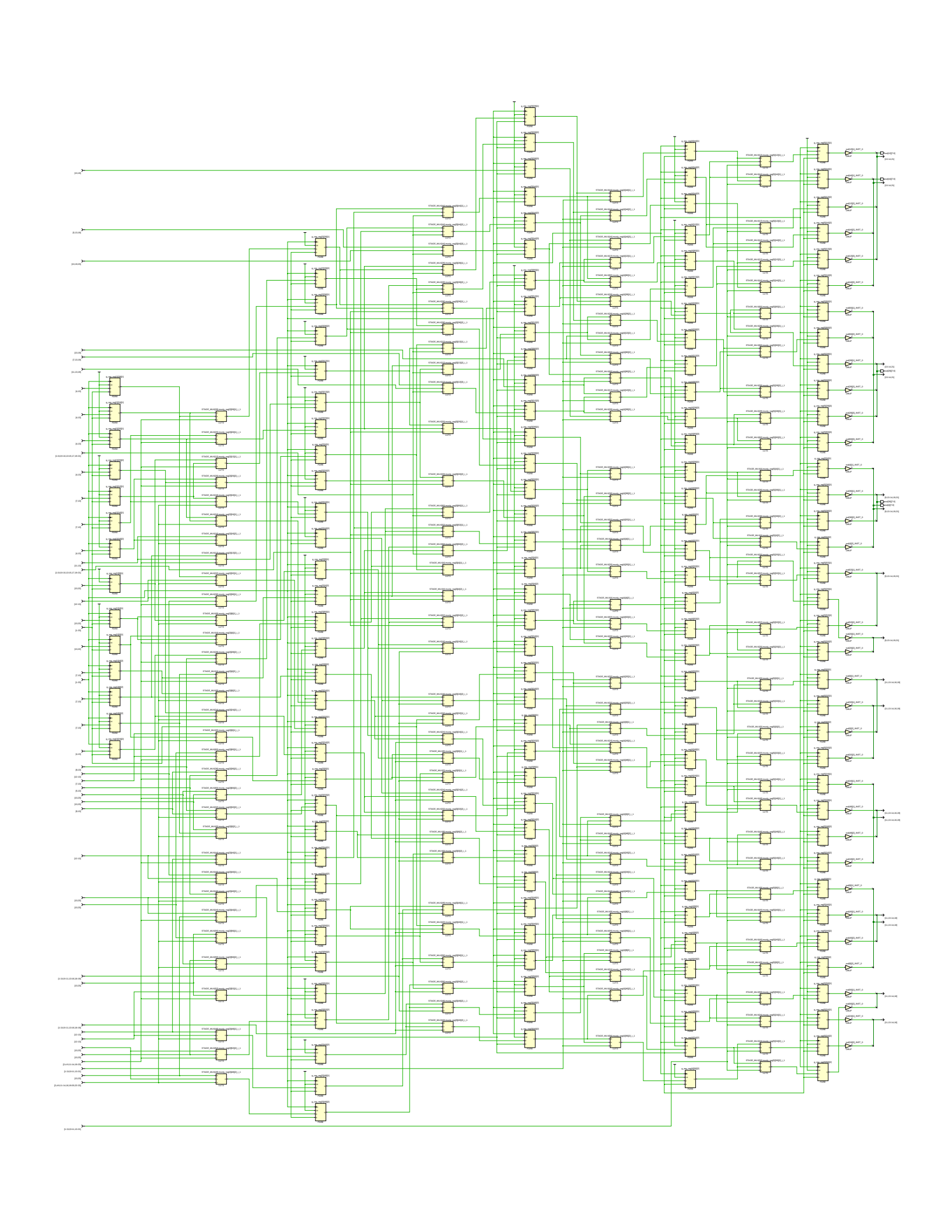


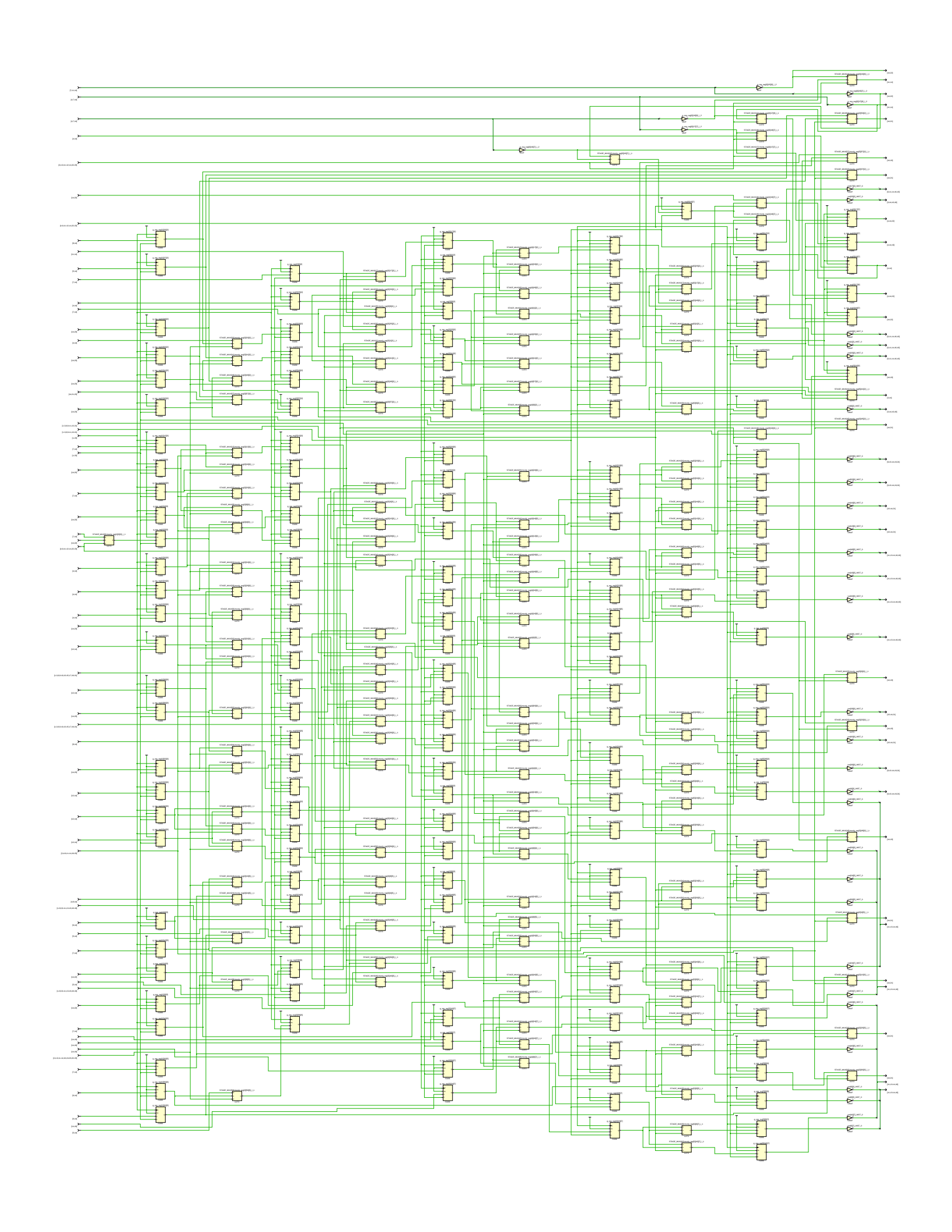


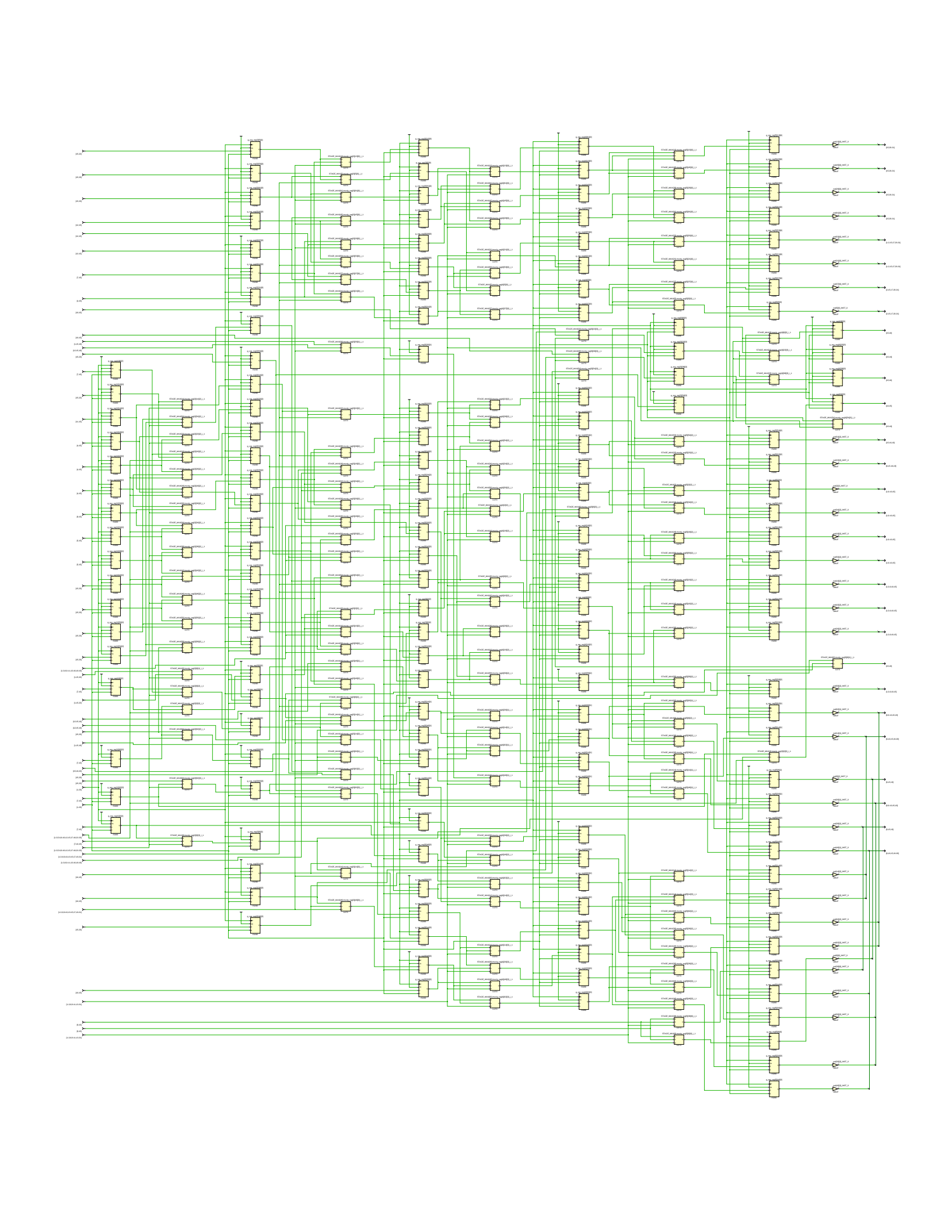


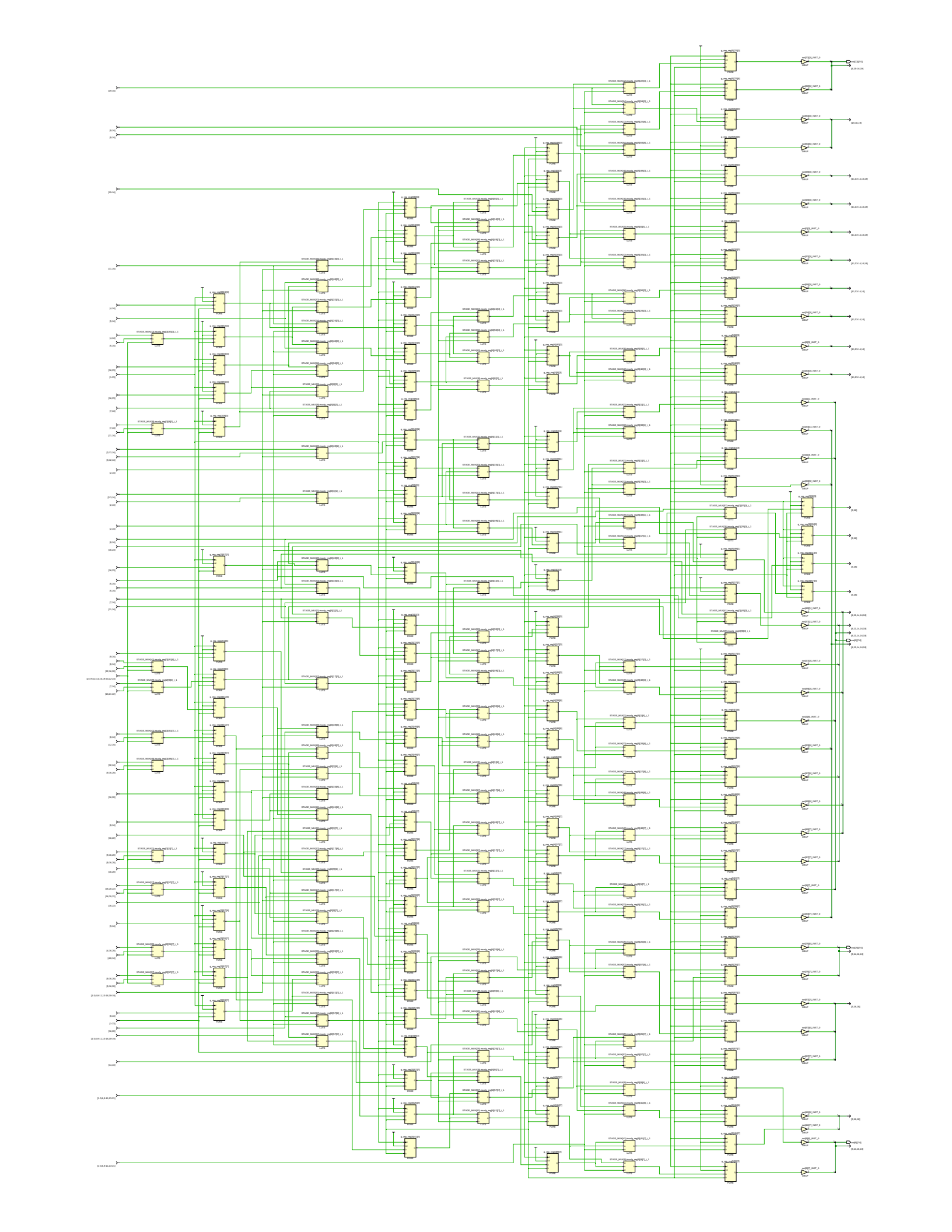


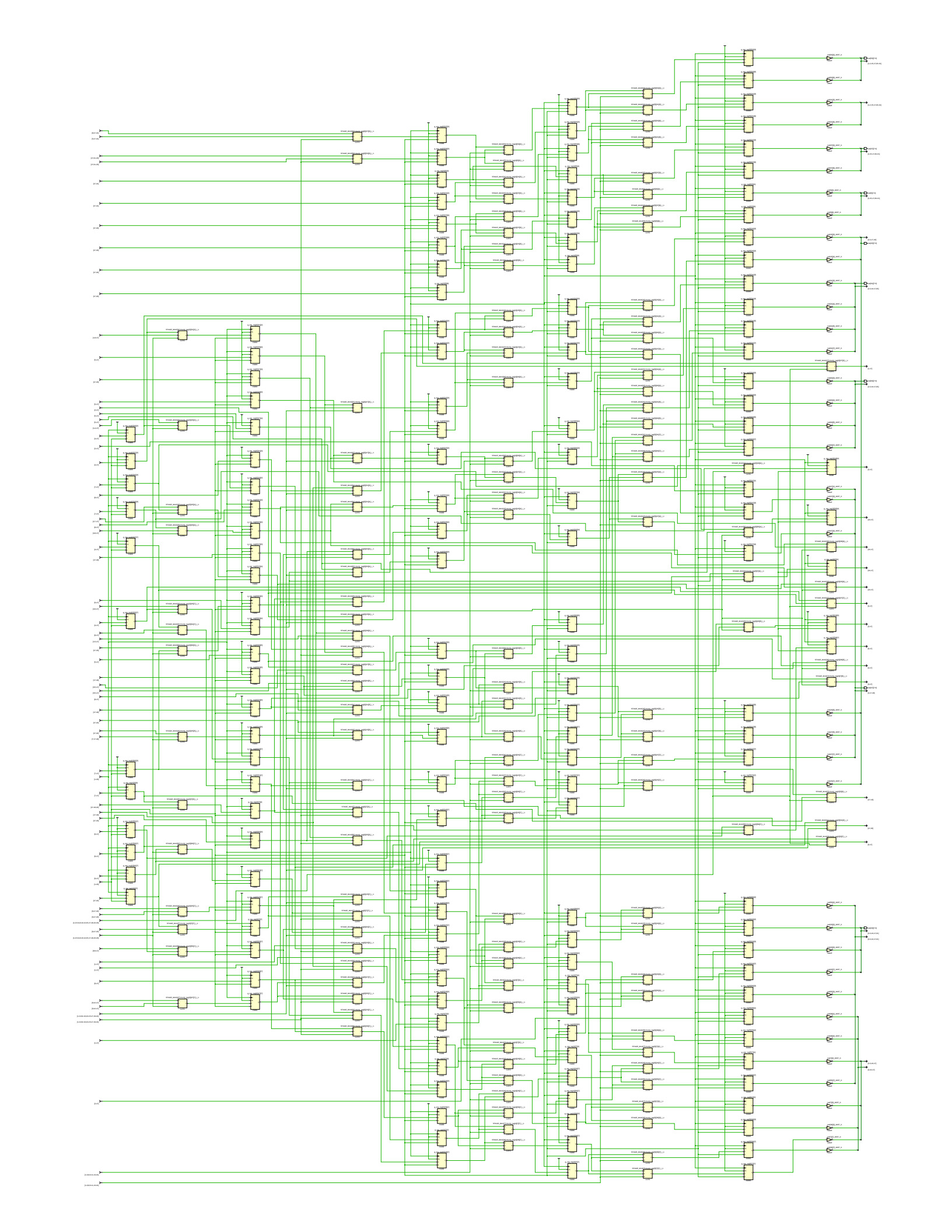


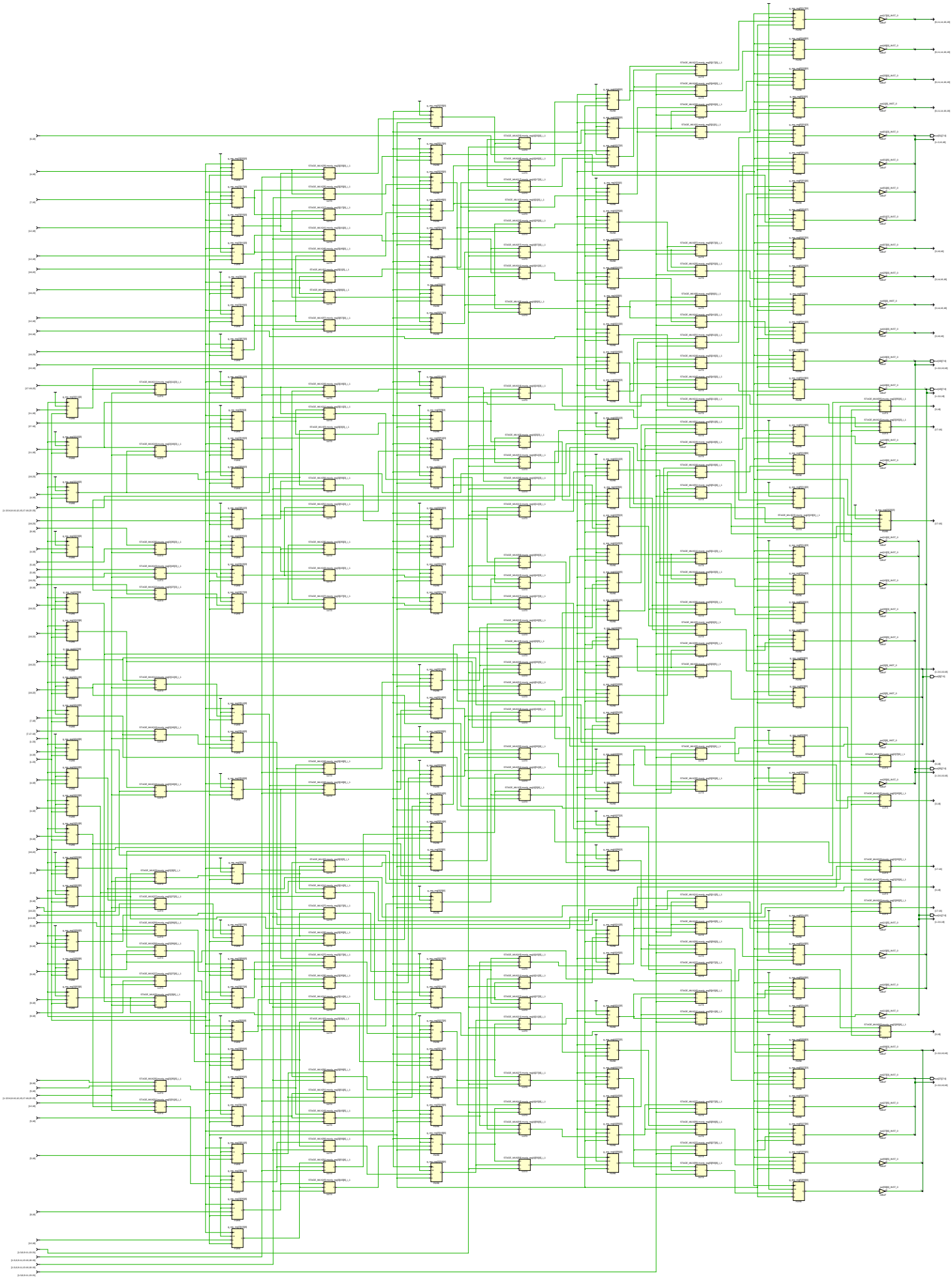


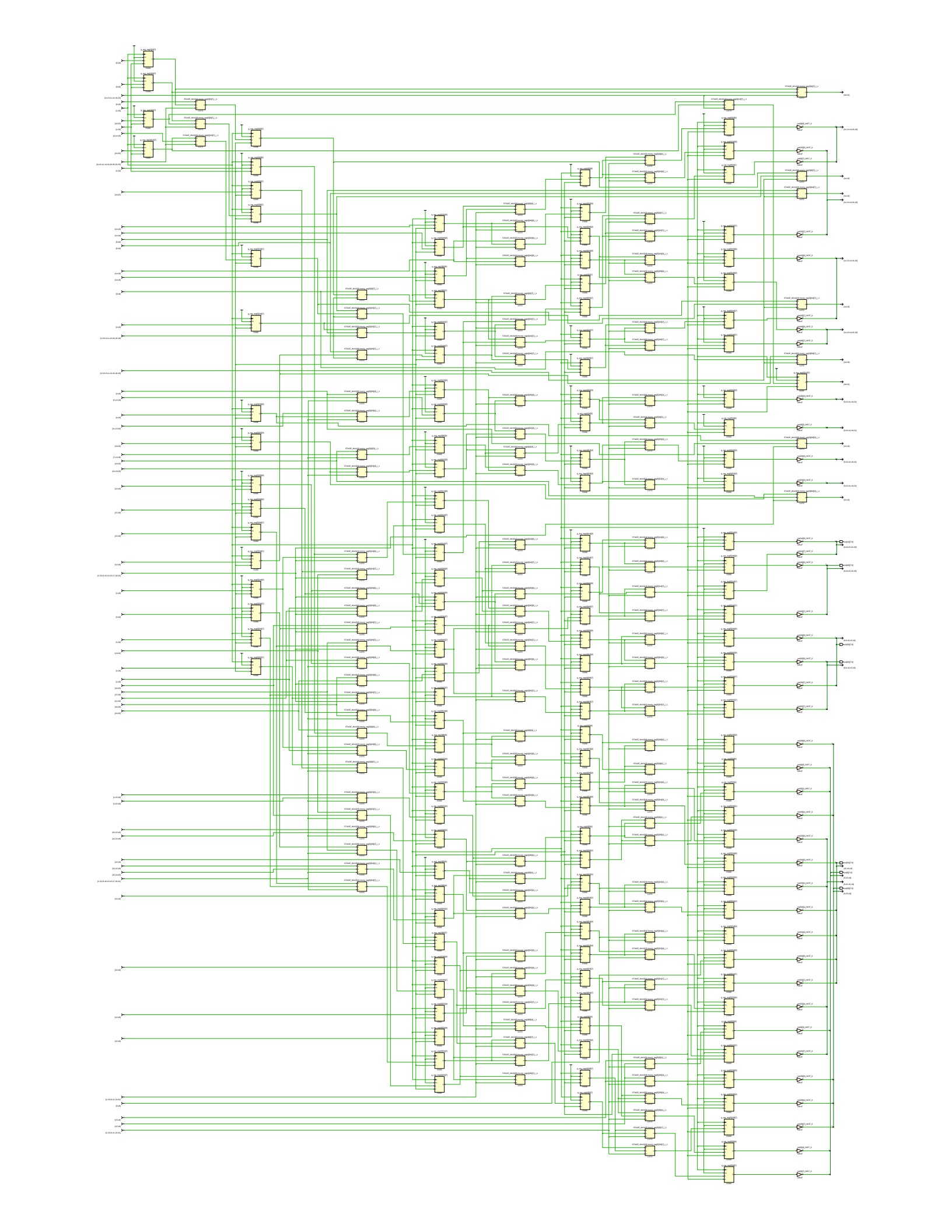


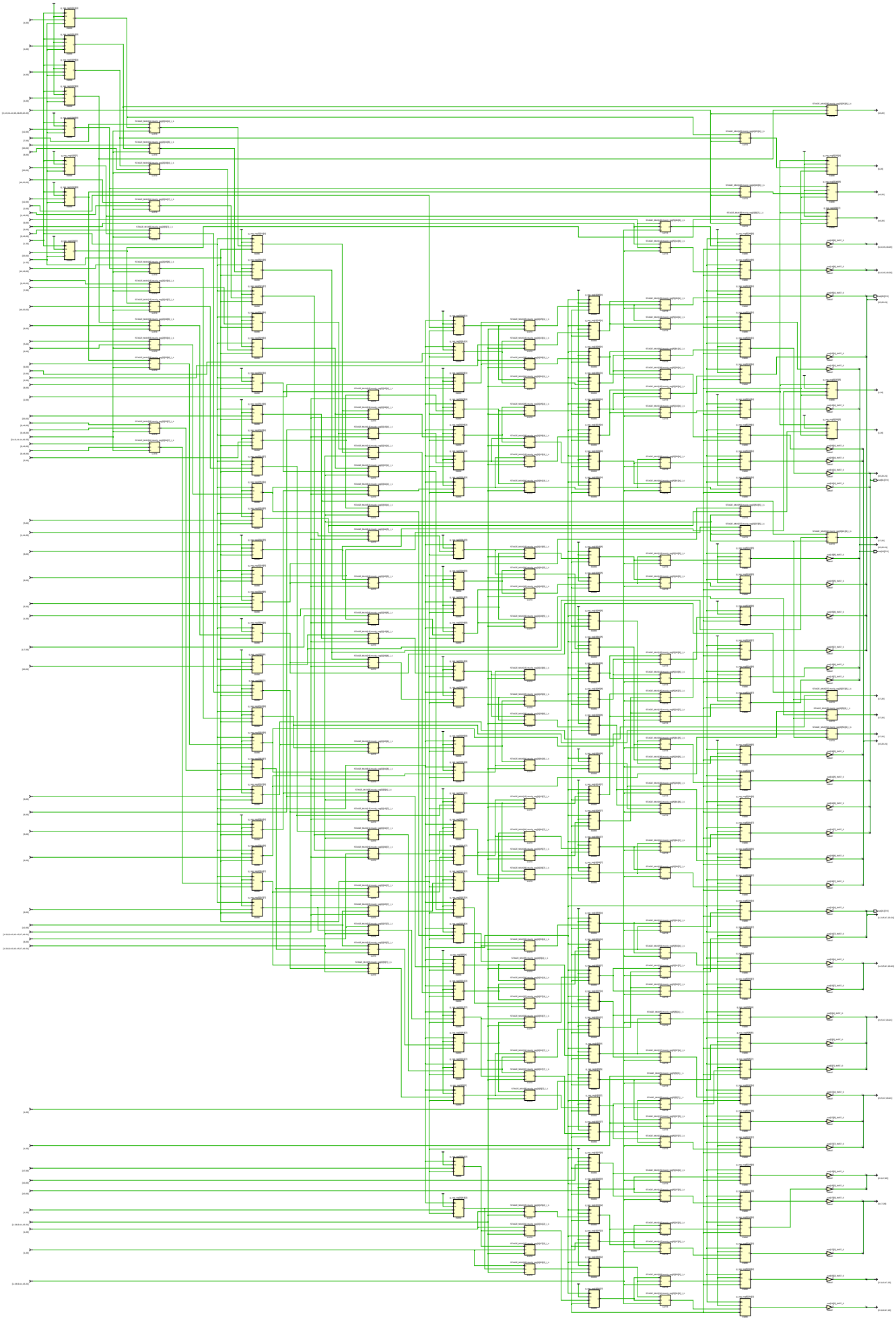


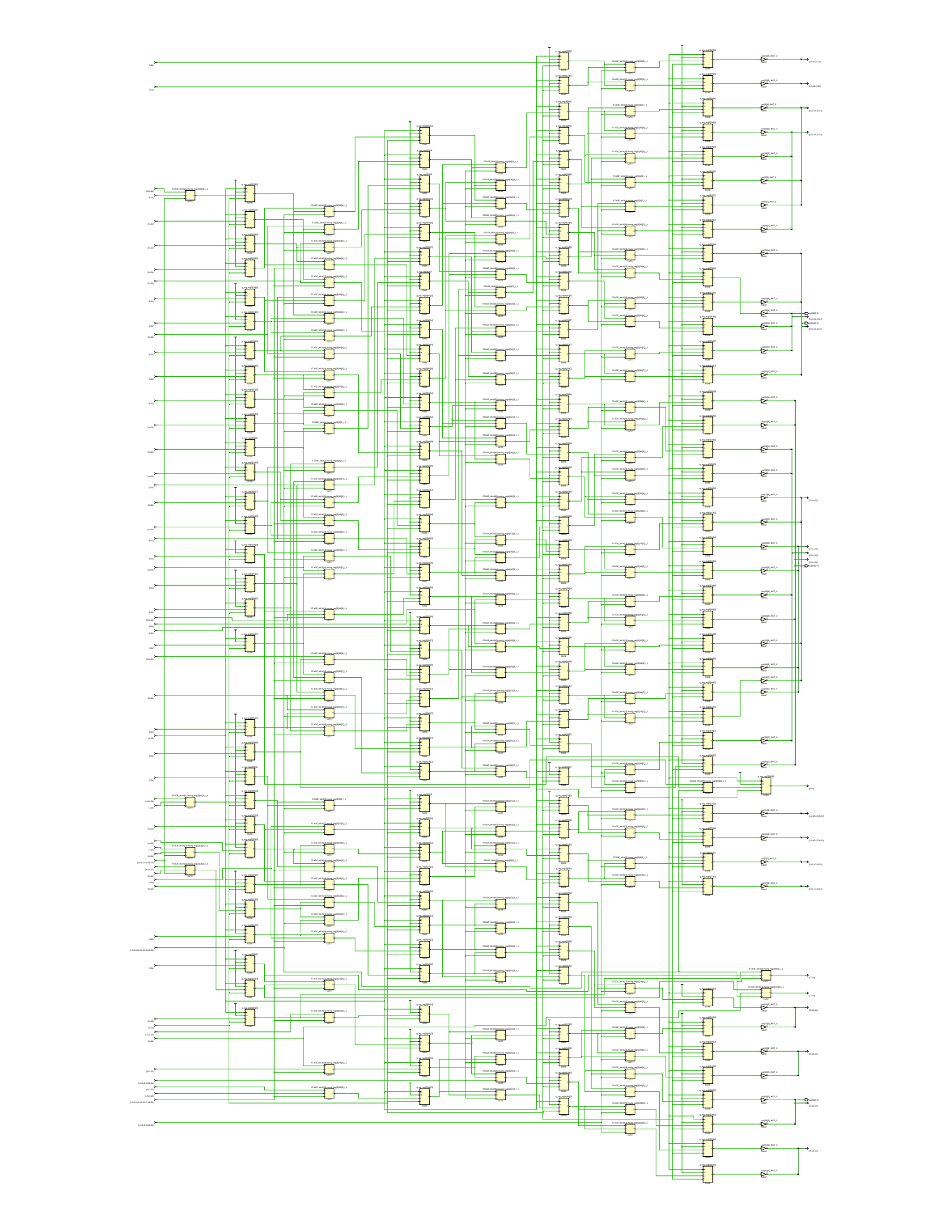


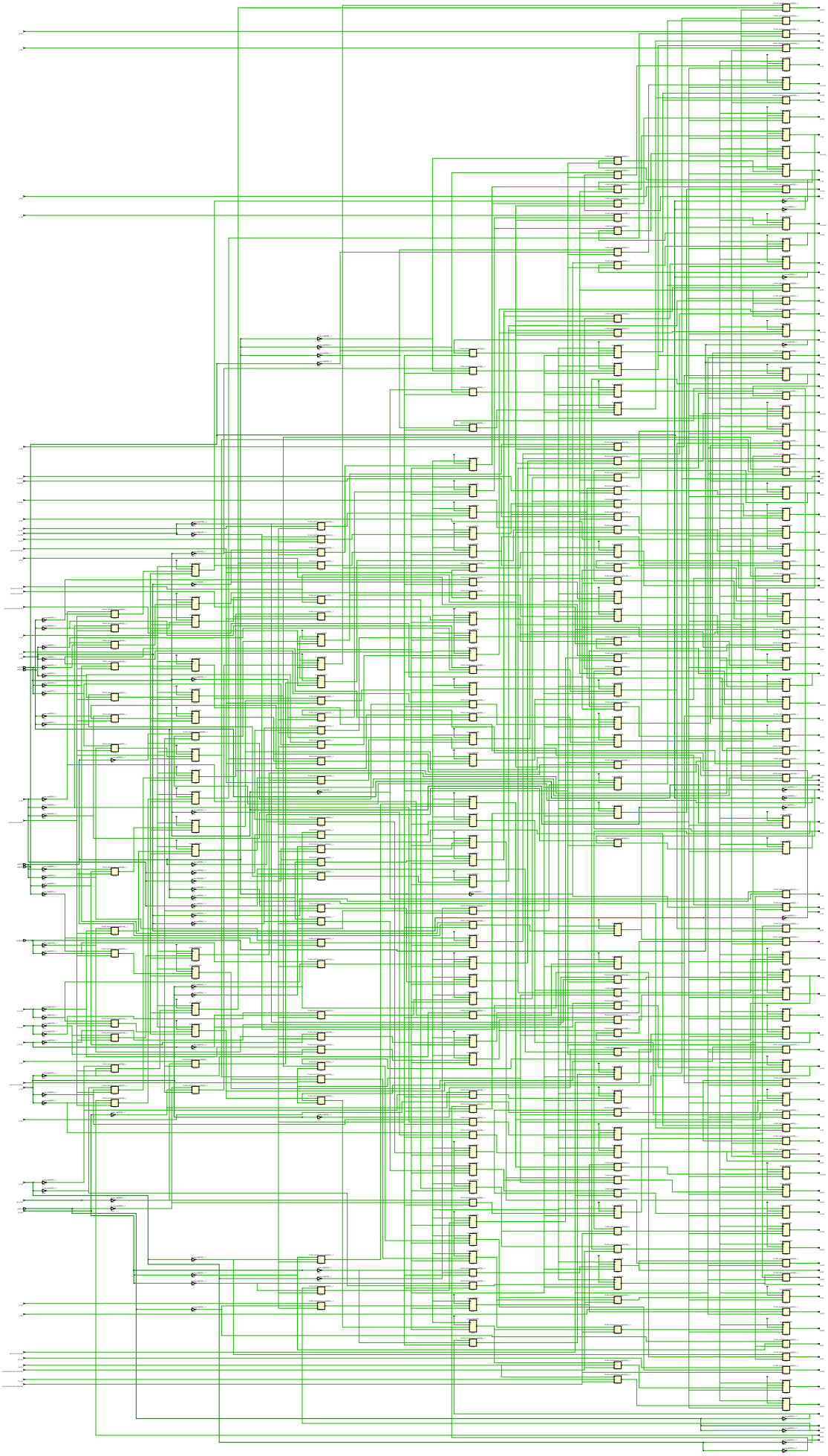


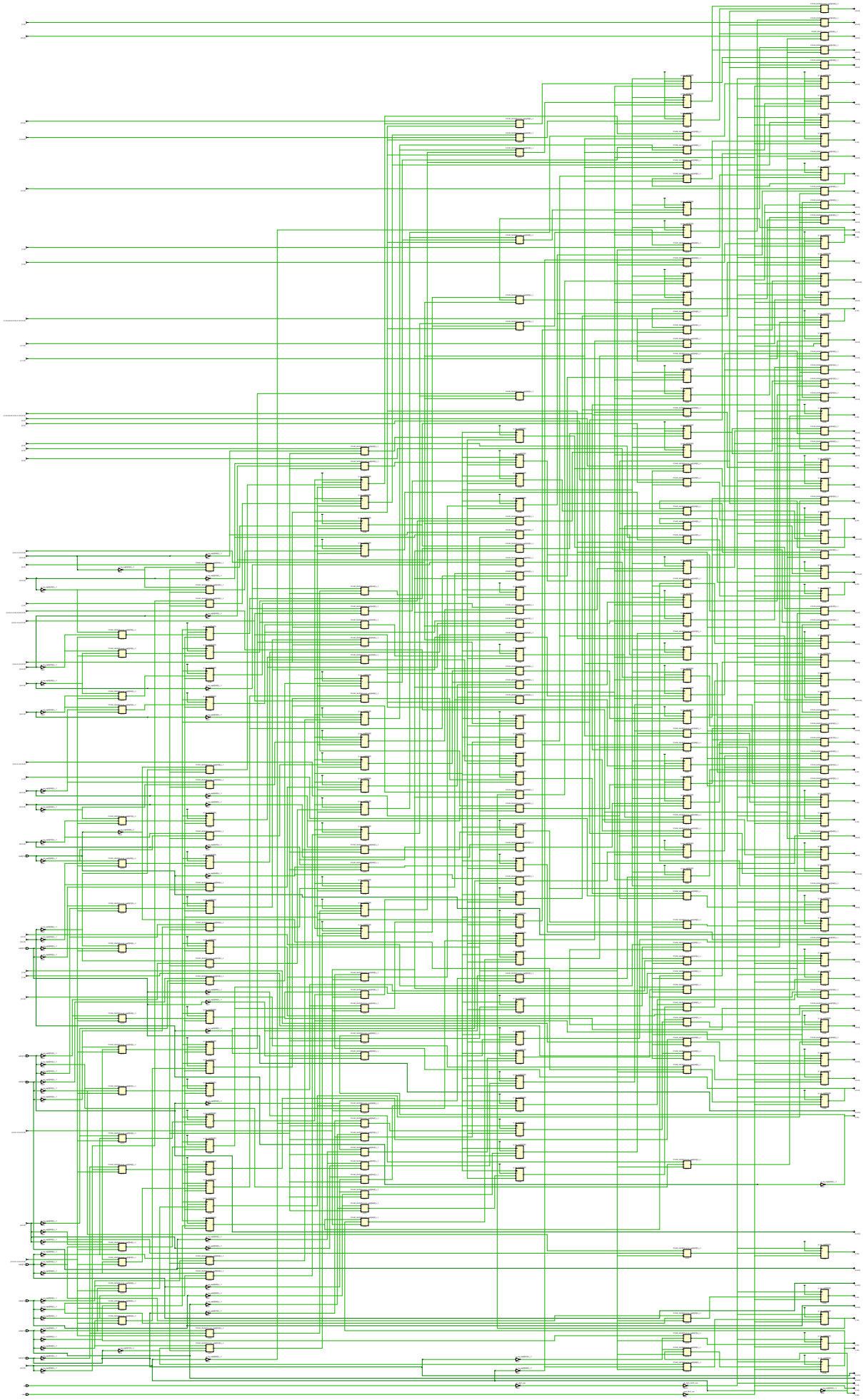








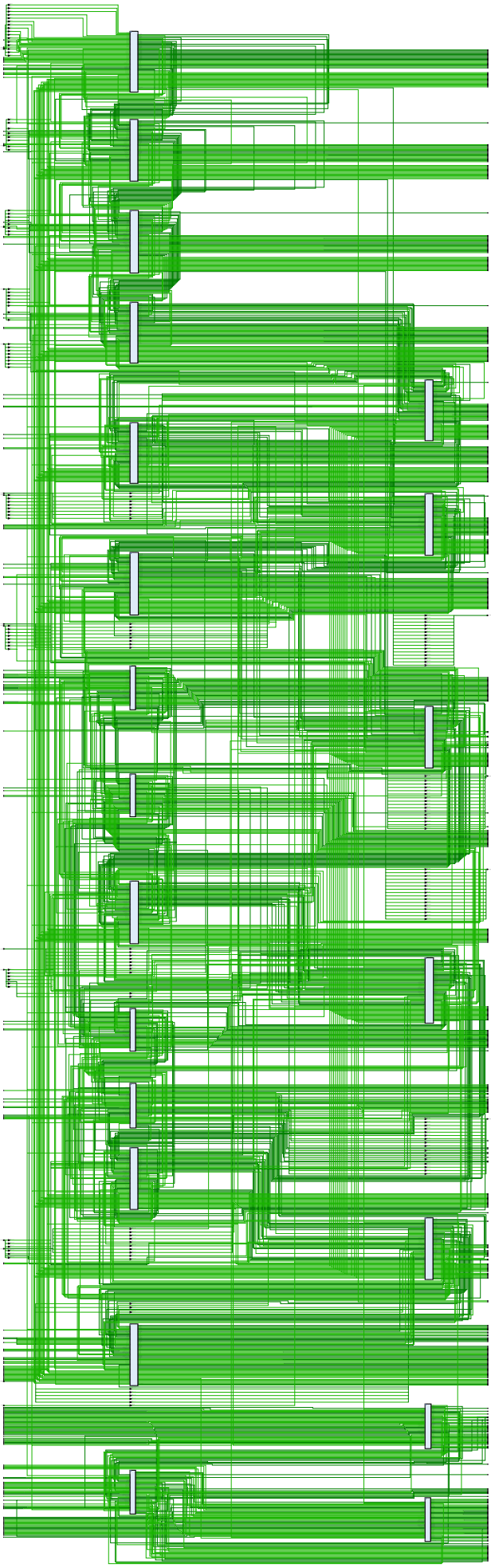


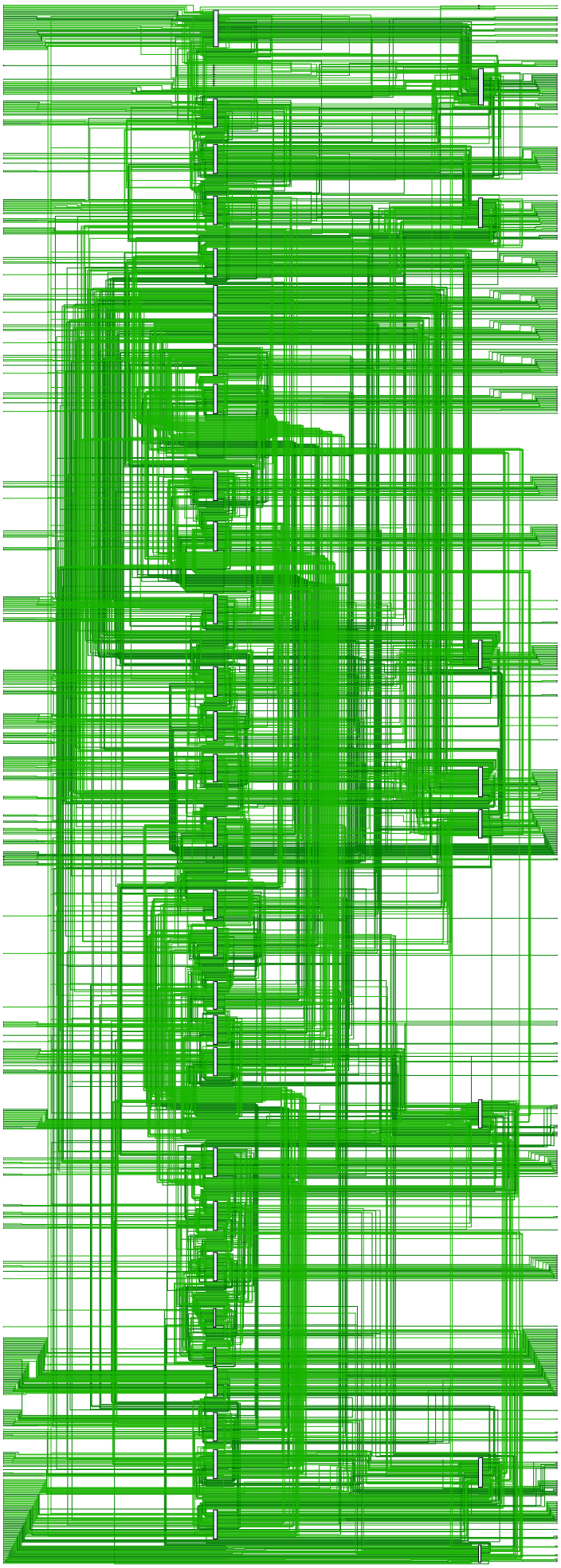


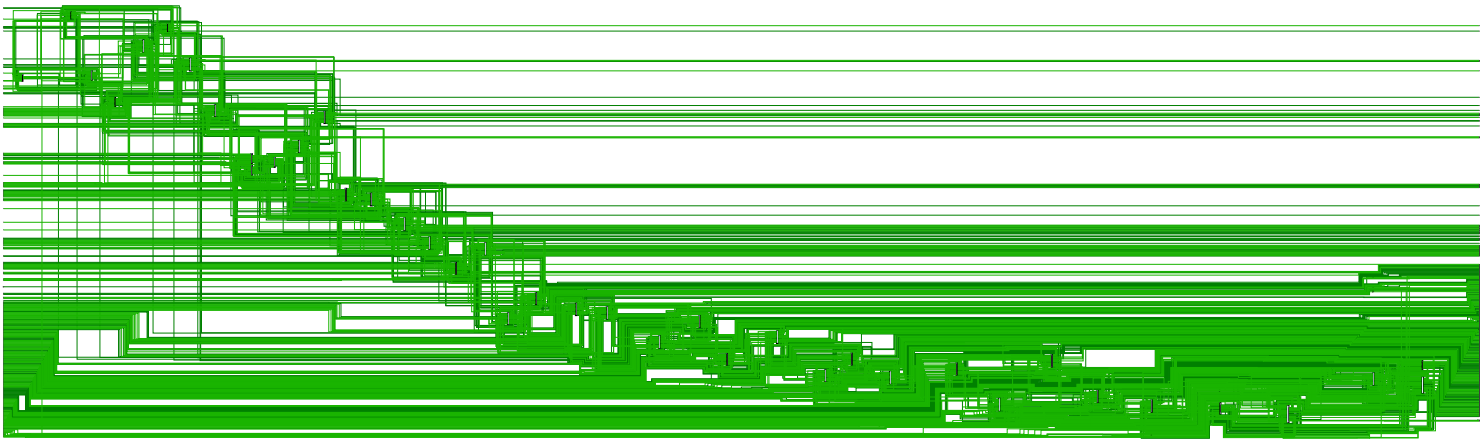
Appendix B: Systolic Array - **16x16** Matrix Multiplication

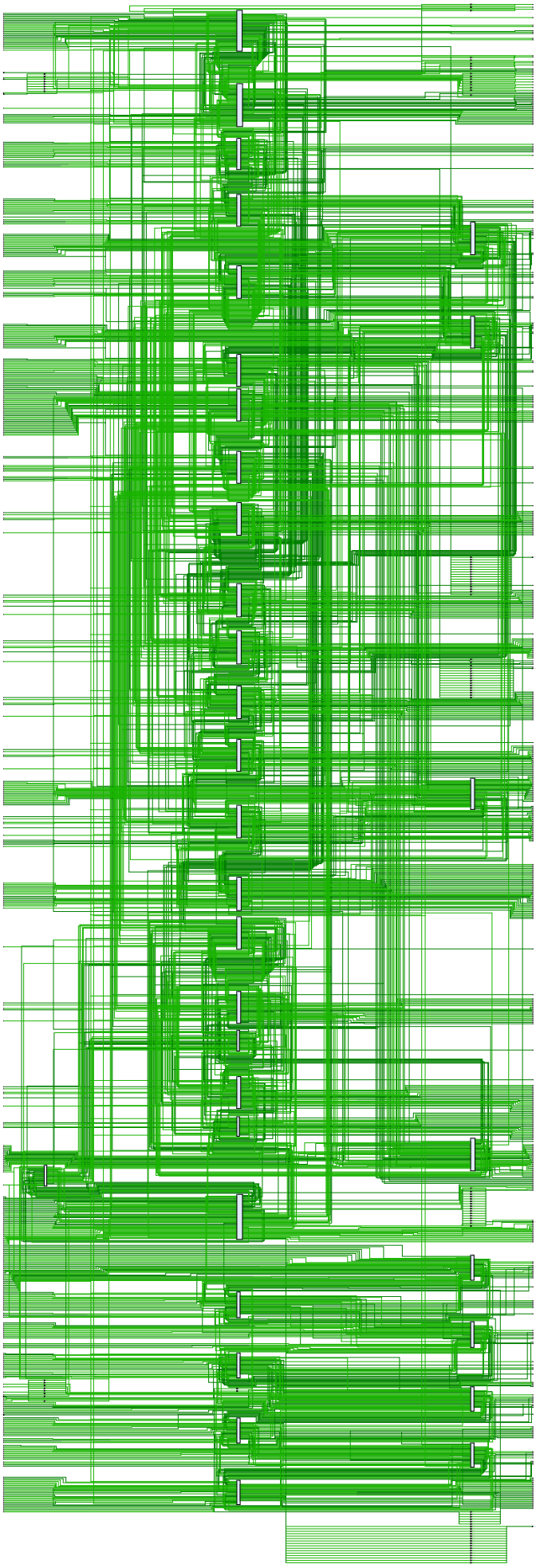
Synthesized Schematic

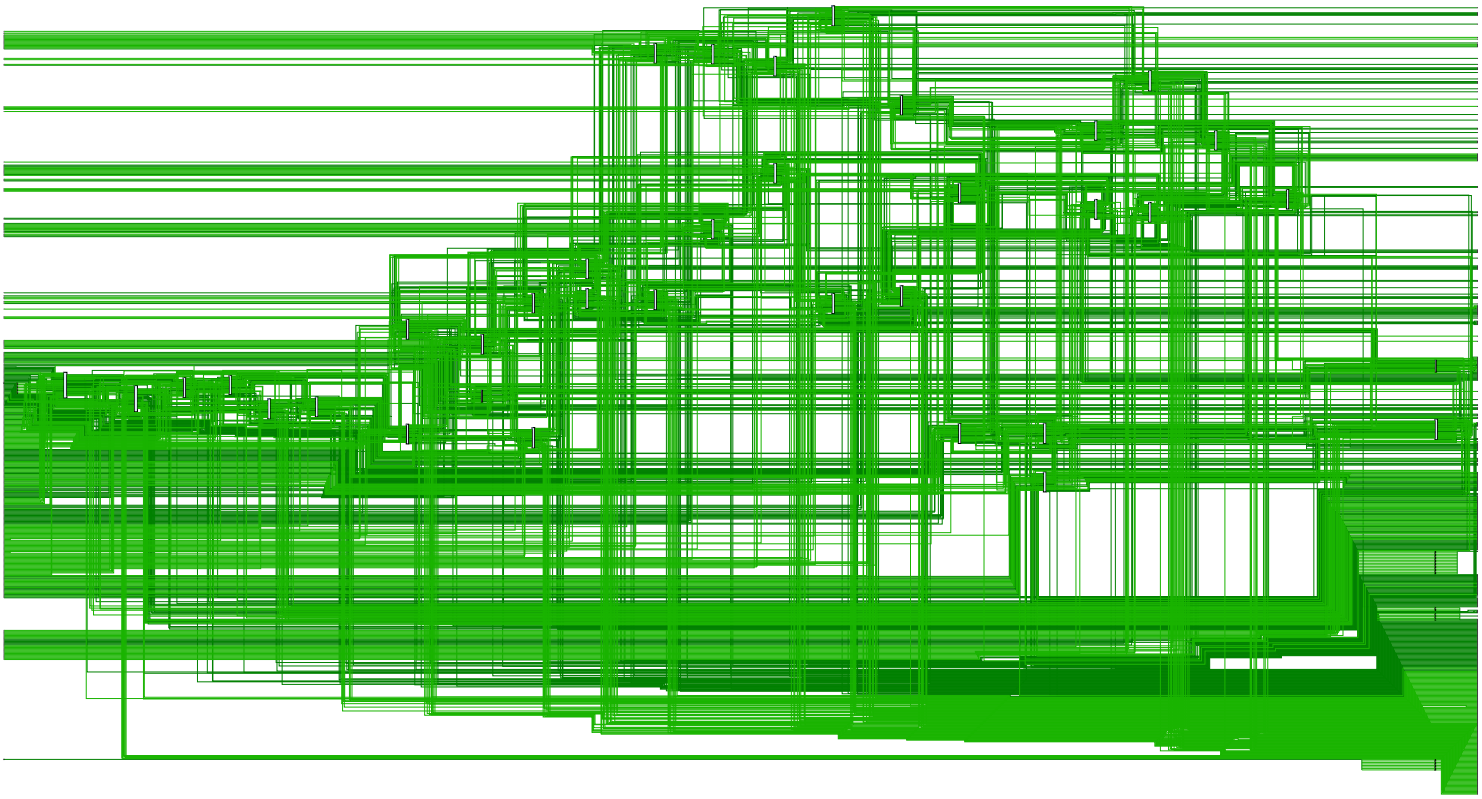
(Inserted in the the next page)

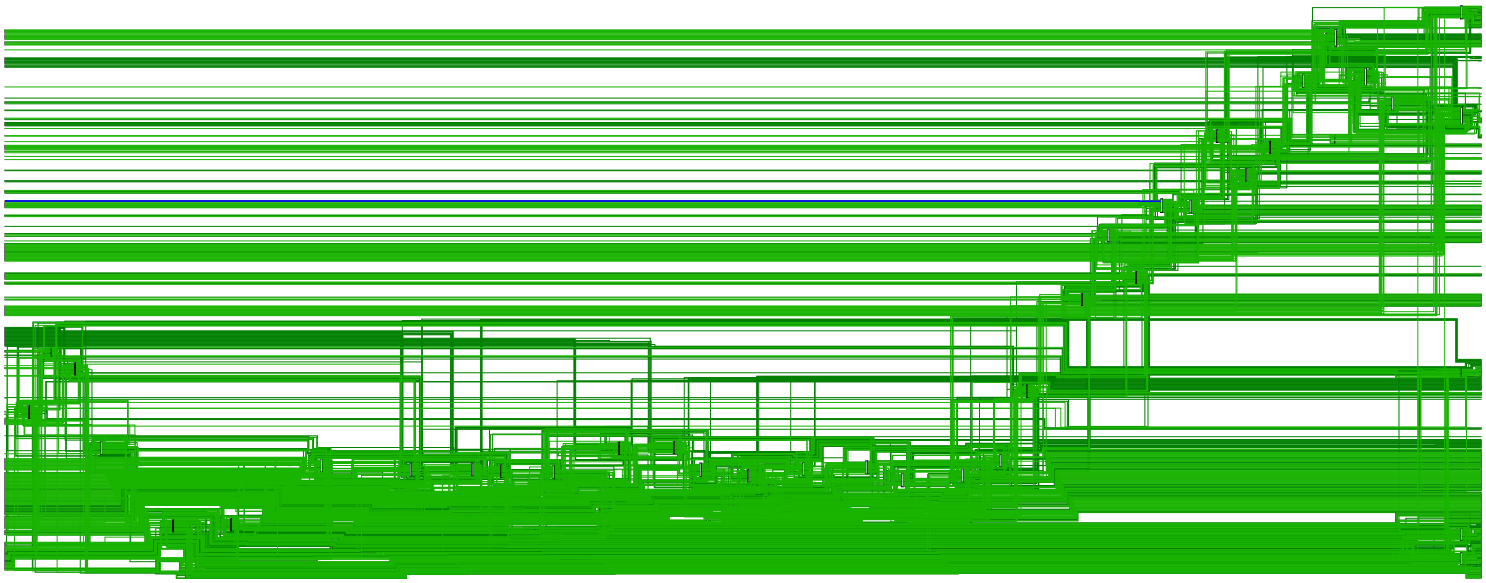


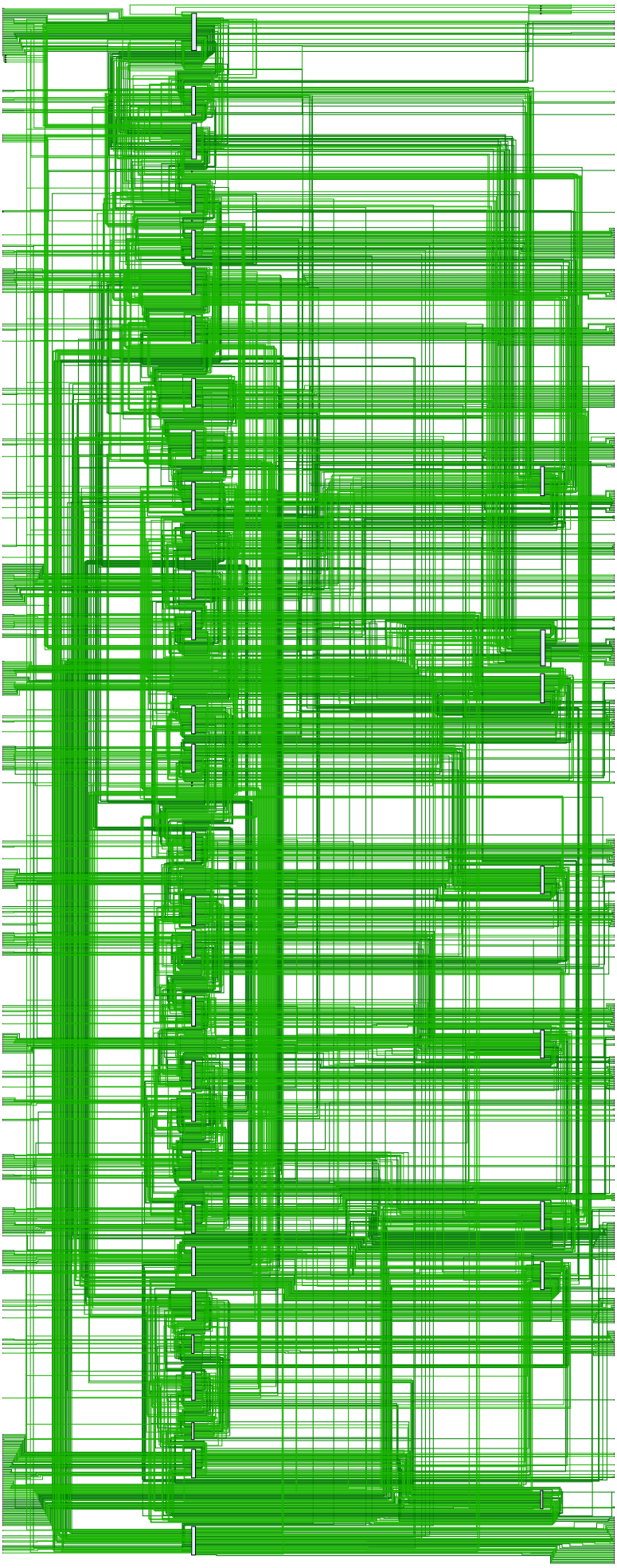


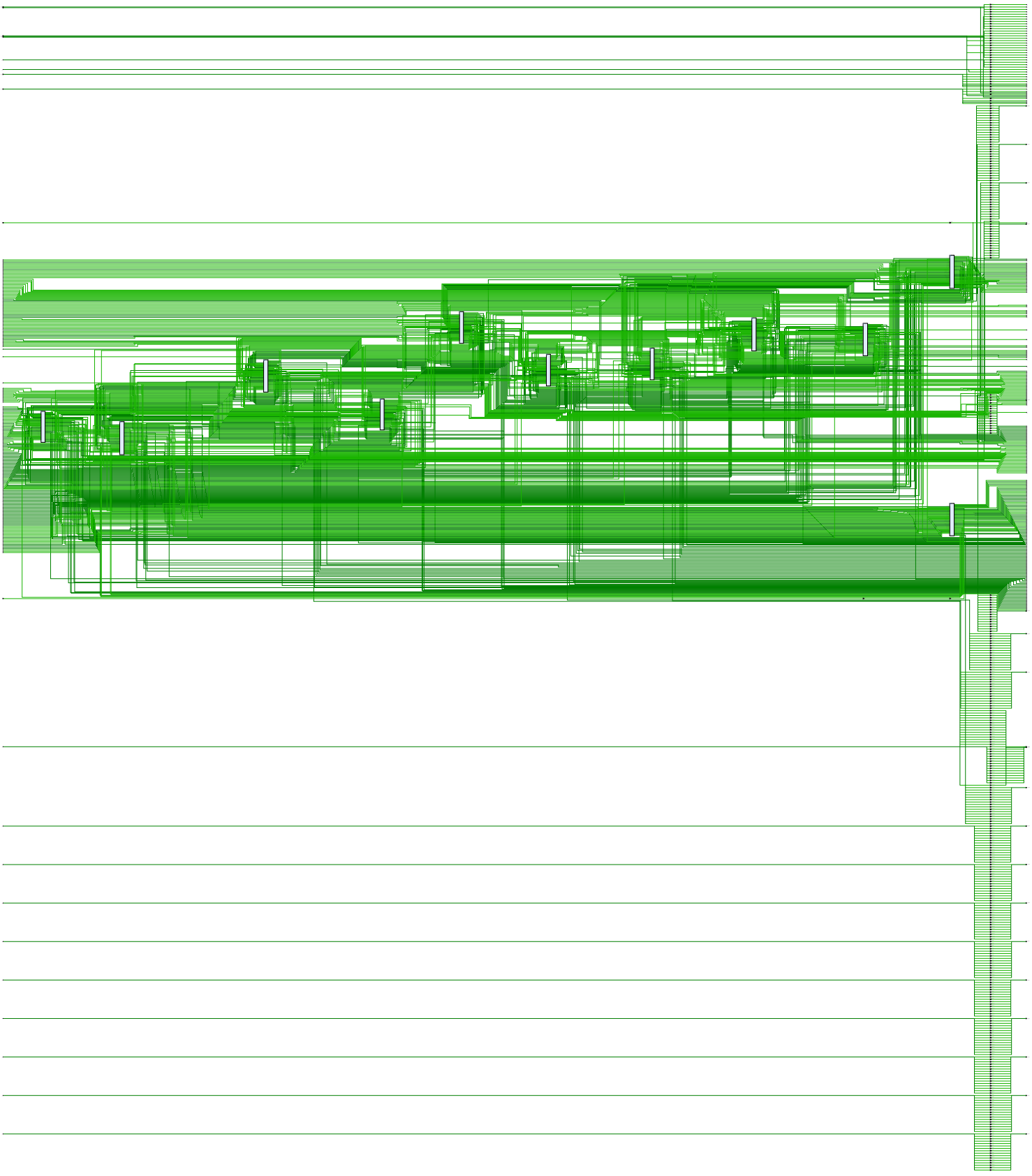












Appendix C: Systolic Array - **32x32** Matrix Multiplication

Synthesized Schematic

(Inserted in the the next page)