European Commission

FVLLMONTI

Call: **H2020-FETPROACT-2020-01**

Grant Agreement no. **101016776**

Deliverable D4.4

Scaled-down N^2C^2 design

Start date of the project: 1st February 2021
Duration: 60 months
Project Coordinator: Cristell MANEUX - University of Bordeaux
Contact: Cristell MANEUX - cristell.maneux@ims-bordeaux.fr

DOCUMENT CLASSIFICATION

Title	Scaled-down N^2C^2 design
Deliverable	D4.4
Estimated Delivery	30/11/2023 (M33+2)
Date of Delivery Foreseen	15/12/2023 (M33+2)
Actual Date of Delivery	12/12/2023 (M33+2)
Authors	Oskar Baumgartner – P4 – GTS Alberto Bosio – P3 – ECL-INL Bastien Deveautour – P3 – ECL-INL Christoph Lenz – P4 – GTS Sara Mannaa – P3 – ECL-INL Cédric Marchand – P3 – ECL-INL **Ian O'Connor – P3 – ECL-INL**
Approver	Cristell Maneux – P1 – UBx
Work package	WP4
Dissemination	PU (Public)
Version	V0.4
Doc ID Code	D4.4_FVLLMONTI_P3-ECL-20231130
Keywords	Compute cube, Logic synthesis, Physical design, Place and route, DTCO methods and tools

DOCUMENT HISTORY

Version status	V0.4
Date	12/12/2023
Document revision	V0.1 – 17/11/2023 – Initial draft V0.2 – 22/11/2023 – Updated Place&Route section V0.3 – 24/11/2023 – Updated N^2C^2 design section (scalable block, logic cell library) V0.4 – 12/12/2023 – Updated generated N^2C^2 section, final review

DOCUMENT ABSTRACT

The aim of this document is to describe the design of a scaled-down (4-bit) Neural Network Compute Cube (N^2C^2). This is a concrete use-case using multiple strands from FVLLMONTI WP3 and WP4 tasks:

- logic synthesis on a "toy" 4-bit N^2C^2 circuit using the the library of JL1 CStatic logic cells described in D4.1 Library of optimized VNWFET-based logic cells with the extracted timing and power characteristics
- development of specific place and route methods and tools for the physical design of N^2C^2 using the intra-N^2C^2 interconnect framework defined in D3.3 3D-place-and-route prototype – V1 and automated layout and GDSII generated JL1 CStatic logic cells described in D4.2 GDSII-ready logic cell and NVM bitcell layout

Overall, this work a) establishes a full toolchain from device to complex logic block and b) enables the gathering of first insights into the actual design, performance and cost of the N^2C^2 block based on vertical technology.

The deliverable is part of work carried out in T.4.3: Configurable N^2C^2 and draws on work described in D3.3 (3D-place-and-route prototype), D4.1 (Library of optimized VNWFET-based logic cells), D4.2 (GDSII-ready logic cell and NVM bitcell layout).

The structure of this deliverable is divided into:

- Logic synthesis of N^2C^2 using JL1 CStatic logic cell library
- Physical design automation (place & route) using synthesized N^2C^2 netlist and JL1 CStatic GDSII layouts
- Closing the loop: post-layout lessons

The information and work described in this deliverable will be used by WP1, WP2 and WP3 to understand system-level implications of technology and device-level characteristics, and by WP5 through the accurate parasitic-enhanced performance estimation of the N^2C^2 block, for use in a systolic array architecture.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016776.

TABLE OF CONTENT

LIST OF FIGURES AND TABLES

LIST OF ACRONYMS / GLOSSARY

1AP: Ambipolar VNWFET technology, single gate stacking
1APFE: Ambipolar Ferroelectric VNWFET technology, single gate stacking
2AP: Ambipolar VNWFET technology, 2-layer gate stacking
2APFE: Ambipolar Ferroelectric VNWFET technology, 2-layer gate stacking
D: Deliverable
CMOS: Complementary Metal Oxide Semiconductor [transistor]
CStatic: CMOS Static [logic]
FeFET: Ferroelectric Field Effect Transistor
GDSII
JL1: Junctionless VNWFET technology, single gate stacking
JL2: Junctionless VNWFET technology, 2-layer gate stacking
JLFE1: Junctionless Ferroelectric VNWFET technology, single gate stacking
JLFE1: Junctionless Ferroelectric VNWFET technology, 2-layer gate stacking
M: Month of the project
MUX: Multiplexer
N^2C^2: Neural Network Compute Cube
P: Partner
P&R: Place and Route
PTL: Pass Transistor Logic
PU: Public
SRAM: Static RAM (Random Access Memory)
V: Version
VNWFET: Vertical Nanowire Field Effect Transistor
WP: Work Package

1. Introduction

The design, physical generation and evaluation of a scaled-down (4-bit) N^2C^2 block is a means of proving the viability of a) the VNWFET technology and b) the overall N^2C^2 concept.

This is a complex endeavor requiring the conjunction of many elements:
- Description of a scalable N^2C^2 circuit design in a synthesizable format
- Liberty library file containing the extracted timing and power data of elementary logic cells to be used in synthesis
- Specific place and route tools for the vertical technology
- .lef file containing the extracted dimensions and pin positions of elementary logic cell layouts to be used in place and route

The aim of this document is thus to explore the means by which a complex function can be synthesized virtually for the emerging VNWFET technology.

The lessons learned from this work and enabling further perspectives are two-fold:
1. to provide a focus point for technology development (WP1, WP2) by moving beyond single device optimization to actual circuit-scale (multiple, interconnected devices) development. Overall, this leads to circuit-scale hardware demonstrations and is intended to prove the viability of the technology.
2. to develop flexible physical design strategies and tools (some from WP3) to pursue realistic technology projection through parasitic extraction on actual physical designed cells, thus enabling more refined performance estimation of the N^2C^2 block for injection into architectural scale explorations in WP5.

This deliverable is organized as follows. We first cover the description of the scaled-down N2C2 design in section 2. We describe how its bandwidth is scalable and analyze the impact of bitwidth on cost (cell count), before detailing the JL1 logic cell library on which experiments are carried out. Section 3 describes the place & route methods and tools developed – the overall flow, the placement method and prototype, and the routing method and implementation. Finally, section 4 applies the developed tools to physical synthesis of the 4-bit N2C2 using the JL1 logic cell library and presents results and lessons learned.

2. Scaled-down N²C² design

In this section, we discuss the design of a scaled-down N²C². We first describe the overall functional structure, including the "black-box" input/output view as well as the essential components. We then discuss the scalability of the N2C2 block, with a view in this deliverable to focus on a scaled-down (4-bit) version of the block in order to test the end-to-end synthesis methodology and toolflow, as well as to gather first insights as to key performance indicators for the approach. A description of the data and logic cell library file then follows, enabling subsequent synthesis and testing for several cases of interest.

I. PREAMBLE

The processing element that makes up the systolic array is referred to as the Neural Network Compute Cube (N²C²). Its primary role is to perform the multiplication-accumulation function within multiple N²C² units constituting a systolic array. Figure 1 illustrates the N²C² block, showcasing its inputs, outputs, and sub-blocks.

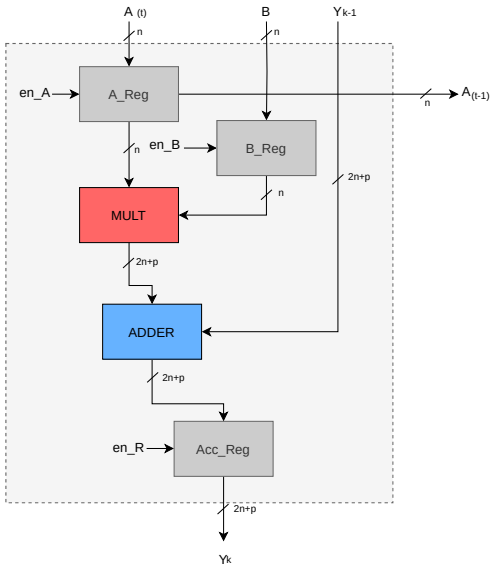


Figure 1: N²C² structure and interconnections

1. *N²C² Inputs*

The N²C² block includes three main input buses: A, B, and Y. Bus A carries the operand for the multiplication, while bus B conveys the weights for this operation. Bus Y carries the accumulation from a preceding N²C² block. Additionally, there are three enable signals (en_A, en_B, and en_R), corresponding to each register within the N²C². These enable signals primarily function as control signals to coordinate the N²C² blocks within the systolic array.

2. *N²C² Outputs*

The N²C² block has two output buses: Bus A (at time t+1 relative to input A) and bus Y. Bus A transmits the operand to the subsequent N²C² block in the current row, while bus Y carries the outcome of the multiplication and addition (accumulation) to the next N²C² block in the current column.

3. *N²C² Sub-blocks*

The first sub-block that stands out is the Multiplier, which receives the 4-bit inputs Operand A and Weight B that were previously stored in their respective register. The Multiplier produces an 8-bit result, as illustrated in Figure 2. To this result, 2 guard bits are appended (MSB side) amounting to a 10 bits output. Depending on the multiplication result's sign, the guard bits will take on the values "00" for a positive result or "11" for negative values, reflecting the Two's Complement representation of negative numbers.

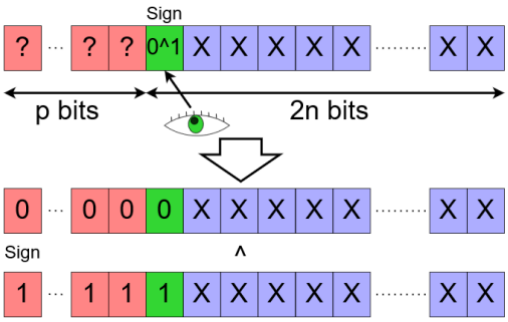


Figure 2: Signed guard bits concatenation

The subsequent sub-block is the adder, which takes as operands the product of the current N²C² block's multiplication and the accumulation result from the preceding N²C² block in the same column. The outcome, consisting of 10 bits, is then stored in the accumulation register, ready for the next clock cycle to transmit it to the subsequent N²C² block.

II. PRINCIPLE OF SCALABLE N²C² BLOCK

The scalability of the N²C² block is examined from two perspectives: firstly, configurability for parallel computing at the expense of precision loss; and secondly, assessing the impact of employing the VNWFET library on both latency and power consumption.

1. *Adjustable bandwidth*

The N²C² block can be used for more than one multiplication accumulation function. For the same amount of clock cycles, the N²C² is capable of splitting its data width into a number of smaller data. In this way, the N²C² can parallelize several computations. Of course, smaller data width will affect the quality of the computation results, since the range of data representation is diminished.

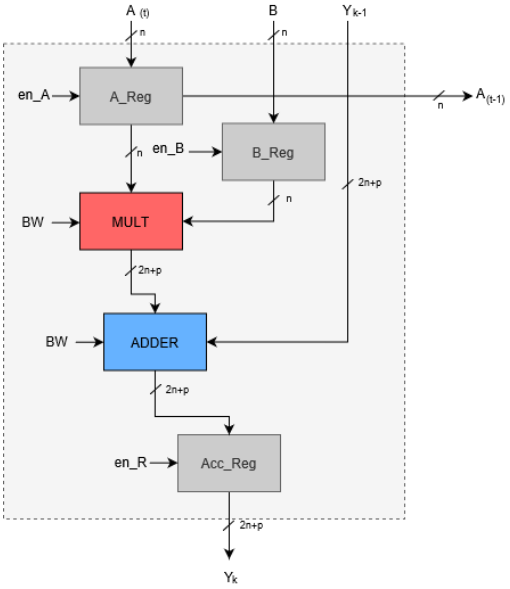


Figure 3: N²C² with BW signal for parallel computing

Figure 4 shows a 32-bit version of the N²C² including an additional control signal (BW) in order to define how input data should be handled, as summarized in Table 1. For each value of BW, the data can be interpreted as 1x32 bits, 2x16 bits, 4x8bits and 8x4 bits.

Table 1: BW signal for data size configuration

BW value	Operands Range (n)	Size of guard bits (p)	Number of parallel computations
00	32 bits	8 bits	1
01	2x16 bits	2x4 bits	2
10	4x8 bits	4x2 bits	3
11	8x4 bits	8x1 bit	4

This approach lends flexibility and throughput to the N²C² in order to improve energy-efficiency at the expense of precision, which can be an advantage for applications that do not handle safety-critical data and are not too impacted by data precision loss. Nevertheless, there is a computing cost overhead since the Multiplier and Adder sub-blocks must include additional logic to supply hardware mechanisms to handle the multiple precision cases.

2. *Scalability for cost analysis*

This scalability serves the sole purpose of measuring the impact of the VNWFET synthesis for different N²C² block sizes. The objective of the scalability analysis is to identify trends and to check if the VNWFET devices can support high-density implementations. The designs are simply four N²C² blocks with the same architecture as shown in Figure 1 with n = 4, 8, 16 and 32 bits respectively.

III. LOGIC CELL LIBRARY

1. *JL1 standard logic cells*

Toward the generation of the desired logic cell library, our work started by the design and characterization of four basic logic cells that should be available in such a library: INV1X1_CStatic_JL1, NAND2X1_CStatic_JL1, NOR2X1_CStatic_JL1and XOR2X1_CStatic_JL1 where the formalism OPnXk_Style_Technology indicates the Boolean operation OP, the number of inputs n and the number of outputs k, as well as the logic design style and the technology variant used to implement the cells. Each cell was studied under different drive strengths and represented by a variety of number of NWs used per cell as shown in Table 2. A detailed study of the static and dynamic behavior of these cells based on SPICE simulations using the VNWFET compact model implemented as a Verilog-A executable model [1] is detailed in deliverable **D4.1**.

Table 2 Number of nanowires used per logic cell

Logic Cell	n-type NW values	p-type NW values
INV1X1_CStatic_JL1	4, 24, 44, 64	4, 24, 44, 64
NAND2X1_CStatic_JL1	8, 48, 88, 128	4, 24, 44, 64
NOR2X1_CStatic_JL1	4, 24, 44, 64	8, 48, 88, 128
XOR2X1_CStatic_JL1	8, 48, 88, 128	8, 48, 88, 128

In addition to these logic cells, we also designed a sequential logic cell (D Flip Flop: DFF) which is necessary for such a library. The DFF is designed with synchronous set/reset control signals. Unlike other cells, for now there is one single instance (no variation in number of nanowires), due to some convergence limitations in the device compact model. The DFF uses mainly 24-NW logic gates, with 2 4-NW INV1X1_CStatic_JL1 gates used as weak inverters in the bistable circuit elements.

The current version of the logic cell library is generated by Liberate™ (Cadence®). The library is named JLNT and is composed of the following logic cells:

- 4 instances of INV1_1_Static_JL1 corresponding to 4,24,44 and 64 NW named as **Xinvx1**
- 3 instances of NAND2_1_Static_JL1 corresponding to 4,24 and 44 NW named as **Ynand2x1**
- 3 instances of NOR2_1_Static_JL1corresponding to 4,24 and 44 NW named as **Ynor2x1**
- 3 instances of XOR2_1_Static_JL1 corresponding to 4,24 and 44 NW named as **Yxor2x1**
- DFFSR (synchronous reset)

where X ∈ {1, 2, 3, 4} and Y ∈ {1, 2, 3}.

2. *Library file format*

For the generated library, we adopted the standard liberty file format. This file format is able to contain timing and physical information about the target standard cells in the library. In our approach, we also adopted the nonlinear delay model where the gate delay is affected by the input slew rate and load capacitance, and is not directly proportional to the input transition time. Thus, lookup tables (LUT) should be defined to store the values of timing and power consumption based on defined indices. For example, defining X input transition values and Y C_{load} values will result in an X by Y LUT that contains all the possible combinations between the defined values and is capable of storing cell delay values or energy consumption.

Liberty file structure

The liberty file itself is constructed in a hierarchical way over three levels: the library level, the cell level and the pin level. Each level stores the required information to be used by the synthesis optimization.

- At the library level, all the information related to the delay model, unit attributes (for the timing, voltage, current, resistance, leakage current and capacitive loads), the operating conditions, the high and low slew thresholds for both the inputs and outputs and the lookup tables (LUT) templates must be specified.
- Each logic cell should have a dedicated description at the cell level. This description starts from the cell leakage power and area to the input/output pins.
- At the pin level, for each pin in the logic cell, it is necessary to define its type (input or output), the logic function (for output pins) and input capacitance (for input pins). Then the timing and internal power LUT data should be included based on the effect of each of the input pins as described below.

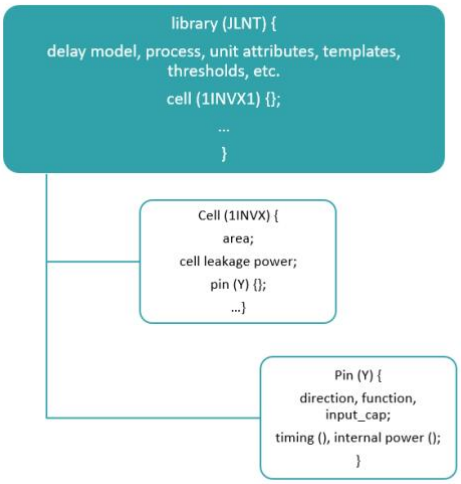


Figure 4 Hierarchy of liberty file

Timing and Power Values

In the liberty file, quantitative descriptions of timing and power consumption data are detailed at the level of the output pins of each cell. For a combinational logic circuit, these matrices reflect the values related to output transitions affected by a single input transition. It is considered here that only one input can trigger one transition at a time, thus leading to output transition. For the correct description of these values, a time sensing parameter should be identified to set how the input transition affects the output transition. It has three possibilities:

- negative unate: falling (resp. rising) input leads to rising (resp. falling) output
- positive unate: falling (resp. rising) input leads to falling (resp. rising) output
- non-unate: No generalized unateness relation between inputs and output

Mainly, the timing is composed of the cell delay and rise/fall time of the output while the power consumption is composed of its rise/fall internal energy values. Normally, these values are calculated as the combination of all the defined input transitions and C_{load} values then stored in the LUTs.

However, for sequential cells, and in order to ensure the proper propagation of data, additional LUTs should be present such as those at the input pins level describing the setup (resp. hold) time constraints i.e. the duration that input data must be stable before (resp. after) the triggering clock edge. These LUTs are usually referred to as constraints LUTs and have values for every defined combination of pin transition (i.e. input pin) and related pin transition (i.e. clock).

3. *Enhanced SMT physical synthesis of logic cell library*

The liberty file should also contain information about the physical information of its cells. For this aim and in order to finalize the physical layout design which allows us to have the needed area values, we use an open source tool [2]. This tool is based on a satisfiability modulo theories (SMT)-based many-tier VFET standard cell (SDC) synthesis framework that provides minimum-sized cell layouts by performing concurrent P&R of FETs. It generated un-sized layouts of the logic cells that can be viewed by using an Excel viewer.

However, in order to be able to extract the needed physical values, these layouts should be resized according to the technology design rules we have. For that, we wrote a python script that is, starting from the SMT tool's output, capable of the generation of the layout in GDSII file format and with the actual size of the cells based on our specified design rules. This approach is described in more detail in D4.2 GDSII-ready logic cell and NVM bitcell layout.

IV. SYNTHESIS FLOW

The logic synthesis process for the N^2C^2 block and its sub-blocks presents no significant challenges, except for describing a behavioral description of the logic blocks using a description language and providing a logic cell library to the synthesis tool. In this instance, VHDL is used, and the logic cell library specifies the basic logic gates used to build the logic blocks. The resulting N^2C^2 netlist after synthesis is generated in Verilog for the sake of simplicity.

Currently, the VNWFET logic cell library includes 4 basic logic cells (INV1X1, NAND2X1, NOR2X1, and XOR2X1), along with one basic sequential unit (D Flip-Flop with synchronous set and reset control signals). The details provided in the library file enable the assessment of the resulting synthesized circuit based on two key parameters: latency and power consumption. The complete logic synthesis process is shown in Figure 5.

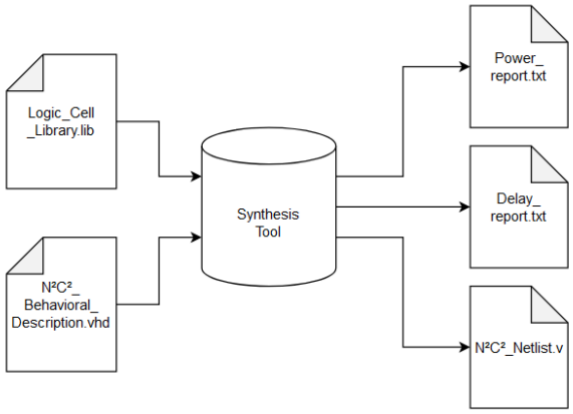


Figure 5: N^2C^2 Logic Synthesis flow

To assess the latency and power that stems from the NVWFET library, the description of the N^2C^2 logic block has to be compared with well-known CMOS logic cell libraries. For this purpose, a 45nm free PDK and a 65nm industrial logic cell library were selected. Both libraries encompass the same four basic logic cells and the D Flip-Flop characterized in the VNWFET library under evaluation. Given that the 45nm and 65nm libraries also include logic cells beyond those in the VNWFET, it is necessary in the interests of making a fair comparison to constrain these libraries to prevent the synthesis tool from utilizing logic cells that could confer them latency and/or power consumption advantages.

V. SYNTHESIS RESULTS AND COMPARISON

Following the flow described in Figure 5, the synthesis of 4 different N^2C^2 blocks of 4, 8 16 and 32 bits is obtained. Each synthesized netlist allows the number of cells that composes them to be counted. Naturally, the cell count is an indication of optimization but does not reflect the footprint since the area of each cell is not yet defined. Nevertheless, the current throughput of a logic cell can affect the cell count. A logic cell with a weak throughput might require the use of two or more of them to handle the same function if the current required at n+1 logic level is too high for a single logic cell.

Table 3 shows the cell count for the 4-, 8-, 16- and 32-bit N^2C^2 blocks when analyzing their netlists with 4 different libraries: the VNWFET library (JLNT), the full 45nm FreePDK library, the constrained 45nm FreePDK library and the constrained 65nm library.

Table 3: Cell count scalability

Cell count	45nm full	JLNT	45nm restricted to VNWFET equivalent cells	65nm restricted to VNWFET equivalent cells
N2C2 4_bits	148	230	530	221
N2C2 8_bits	360	737	1674	705
N2C2 16_bits	972	2553	5697	2321
N2C2 32_bits	2999	9395	20608	8492

These results show that, as expected, a full library greatly diminishes the cell count. Since most of the multiplier and the adder are very regular arithmetic sub-blocks, the use of cells such as the FA(Full Adder) and the HA(Half Adder) naturally reduces the cell count.

Now, examining the results from the VNWFET JLNT library with respect to the 45nm and 65nm constrained libraries proves to be of more interest (and a fairer comparison). VNWFET performs better than the constrained 45nm library. This can be explained by the limited logic cells available for one each function. While the VNWFET has 3 NAND2x1, NOR2X1 and XOR2X1 as well as 4 INV1X1, the 45nm only has one logic cell of each of these functions.

This is not the case for the restrained 65nm, which counts with more throughput than the VNWFET library. This shows in the results where the restrained 65nm library performs slightly better respect to the VNWFET library.

Indeed, the latency reported by the synthesis tool is shown in the graph in Figure 6. From the figure, we can observe that the VNWFET performs poorly in terms of latency when compared to the 65nm restrained library. This might be due to the better throughput option that the 65nm library can offer. In fact, the VNWFET library performs better than the restrained 45nm that has less throughput option.

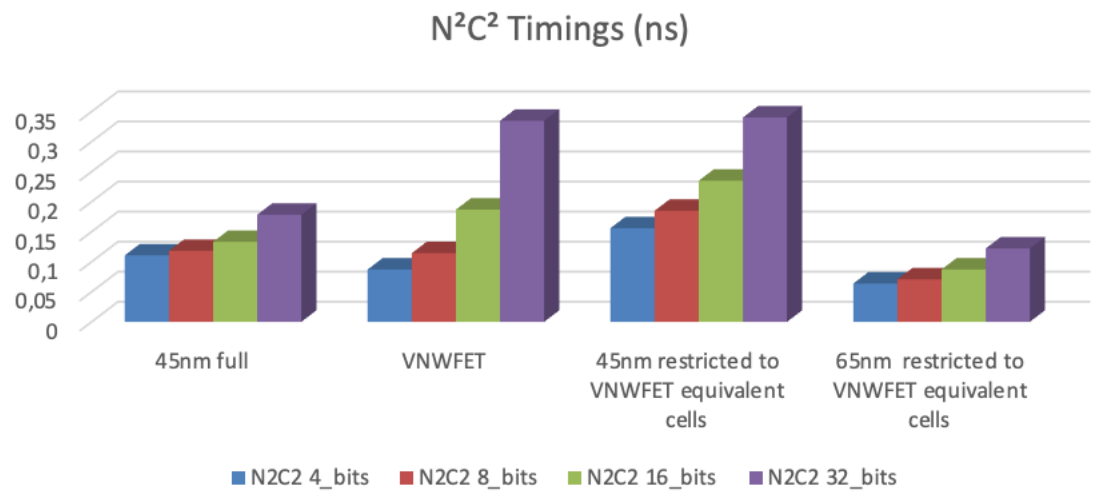


Figure 6: N²C² latency scalability

The power consumption of the N²C² is depicted in Figure 7 and is composed of three different sources of power consumption: internal power, power due to switching activity, and leakage power. The fourth data is the total power, which sums up the previous values. For each N²C² synthesis, we can observe very positive results.

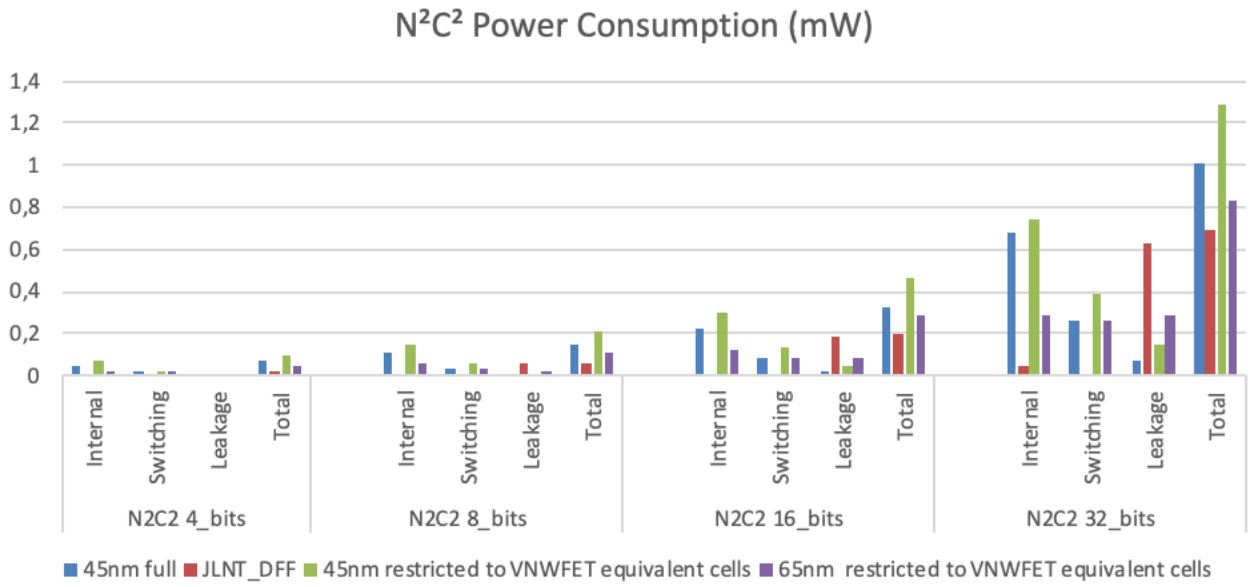


Figure 7: N²C² power consumption scalability

For each version of the N²C², the trend reads equally: both versions of 45nm perform worst, while the restrained 65nm and VNWFET libraries have very similar power consumption. When looking at the details however, we can observe that the leakage power of the VNWFET library represents its highest consumption. This goes against the behavior of the other libraries. The same can be said about the internal power. In that case, the VNWFET performs well with respect to the other libraries. The switching activity has yet to be figured out since its value does not appear for the VNWFET library.

3. Place & Route

The goal of Place & Route is to generate a physical representation of the netlist that results from the logical synthesis (see Section 2).

While several commercially available tools for Place & Route exist, these tools adhere to certain design rules, presenting a lack of adaptability to meet the needs for the N^2C^2. Therefore, an in-house developed Place & Route tool gives us the flexibility to explore and optimize the chips design with respect to placement rules (see I) and the 3D nature of multi-gate VNWFETs. To that end a prototype for a Place & Route tool to generate the physical layout of the N2C2 has been implemented in Python. All results presented in this section were generated with this prototype.

This physical design is carried out in two steps:

1. Generation of the placement of the cells on the chip surface. In our approach, this placement uses heuristics to estimate the expected length of the wires in the netlist, which is used to place the cells such that the expected overall wire length is minimized.
2. Generation of the physical connections (i.e., physical wires) between the cells. Here, we use a maze routing algorithm which is guided by a cost function that generates physical wires for all wires in the netlist.

To generate the physical layout of the N^2C^2 a prototype for a Place & Route tool has been implemented in Python. All results presented in this section were generated with this prototype.

I. PHYSICAL SYNTHESIS FLOW

Generating an optimal placement of cells on the chip surface and creating physical wires with minimal length between the pins of the cells are computationally demanding problems (both problems are NP-hard) [1]. Therefore, it is not feasible to generate a placement by hand and powerful algorithms based on sophisticated heuristics are required. Furthermore, the cells must abide by pre-defined placement rules to ensure the validity of the generated placement.

The placement rules for the N^2C^2 are as follows:

- all cells are placed into rows, while so-called *routing channels* occupy space between the rows.
- the routing channels are primarily used to route physical wires between cells in adjacent rows.
- each cell is 3 metal layers high (M1, M2, M3)
- all pins of the cells face the closest routing channel and are placed on M3.
- the cells are powered by power strips located on M1
- the power strips are supplied by so-called *power elevators*.
- the power elevators must be placed at constant intervals along each cell row.

Figure 8 shows an illustration of the above-mentioned placement rules.

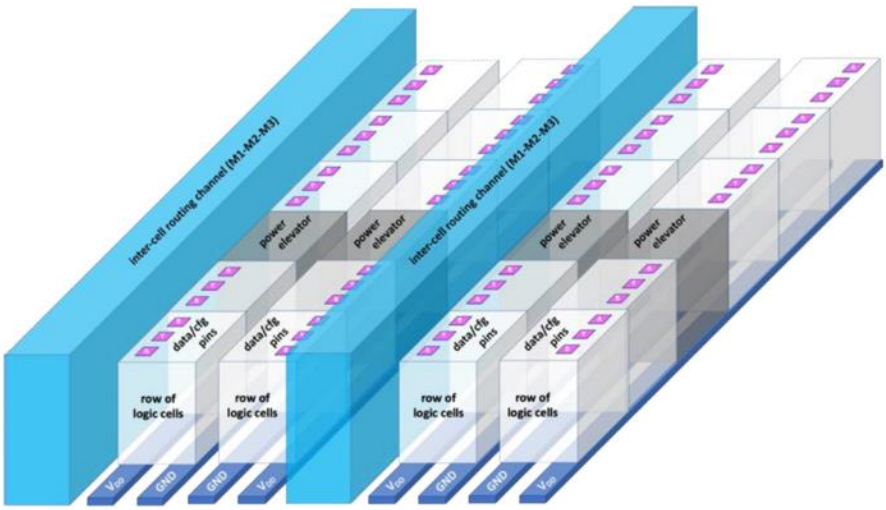


Figure 8 : Illustration of cells and power elevators in cell rows and the routing channel.

Furthermore, the size of the routing channel can be increased depending on the number of expected physical wires. This provides more space for the physical wires on M1-M3 and prevents the creation of additional metal layers.

Another important metric that needs to be optimized is the footprint of the chip. The footprint must have an aspect ratio of approximately 1. Furthermore, the unused space on the chip (i.e., space that is not occupied by cells or wires) must be minimized.

The computation of the footprint begins by calculating the total lengths of all cells (i.e., the size of the cells along the routing channel) that are placed on the chip. In addition, the number of power elevators required in each row is calculated and added to the total length of all cells. Subsequently, the length of all cells is divided by the width of the cells and rounded up – thus creating a strip of rectangles. The number of required rows is then determined by taking the rounded up square root of the number of rectangles. Finally, the side length of the chip is calculated by multiplying the number of rows with the width of the cells and adding additional space for the routing channel.

If one or several placed cells extend beyond the calculated side width (in the x-direction), the footprint size in this direction is increased to fit all cells on the chip. This strategy leads to chips that do not have an exact aspect ratio of 1. However, the error in the aspect ratio is negligible and diminishes with increasing overall footprint.

II. PLACEMENT METHOD AND IMPLEMENTATION

1. *Method and prototype*

The Place & Route prototype uses *simulated annealing* to generate a placement of the cells that aims to minimize the overall physical wire length [4]. The physical wire length of the wires in the netlist is estimated with the so-called *Half Perimeter Wire Length* (HPW).

The HPW is calculated by determining the bounding box (i.e., the minimal rectangle) that contains all pins that a wire links together (see Figure 9). The HPW of a cell is the sum of the HPWs of all wires connected to that cell.

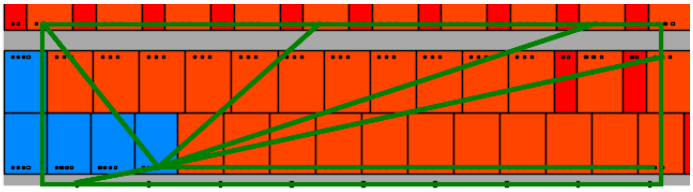


Figure 9 : Illustration of the half perimeter wirelength of a wire with fan out.

Simulated annealing starts by creating a random placement of all cells in the rows of the previously calculated footprint. Afterwards the positions of the cells are randomly swapped and the change in HPW is calculated (Δ_{Length}). If the HPW decreases the swap is accepted otherwise the swap is reversed. This *greedy* approach tends to get stuck in local minima. Therefore, to explore more of the solution space (i.e., different placements) a temperature parameter (T) is introduced. Depending on T and Δ_{Length}, cell swaps that increase the HPW are randomly accepted, which allows the placement to escape local minima and find better placements. Over time the temperature T is reduced (cooling rate) until only improvements in the HPW are accepted. The simulated annealing stops when a predefined number of swaps does not improve the HPW. Figure 10 illustrates the change in the HPW during the simulated annealing.

The final step in the placement process is the positioning of the power elevators. As previously mentioned (see Section I), the power elevators have to be placed in pre-defined intervals along each cell row. If already placed cells interfere with the placement of the power elevator, they are pushed to the side to make space for the power elevator. Figure 11 shows the cell placement of the 4-bit N^2C^2 at each of the 3 stages during the placement process.

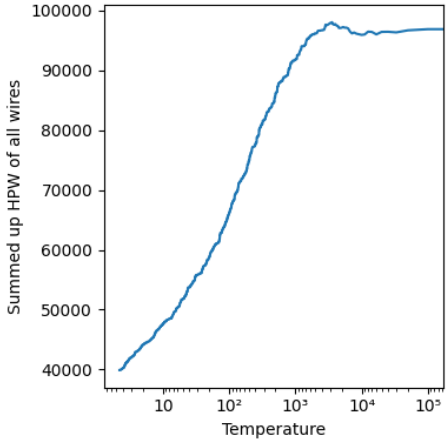
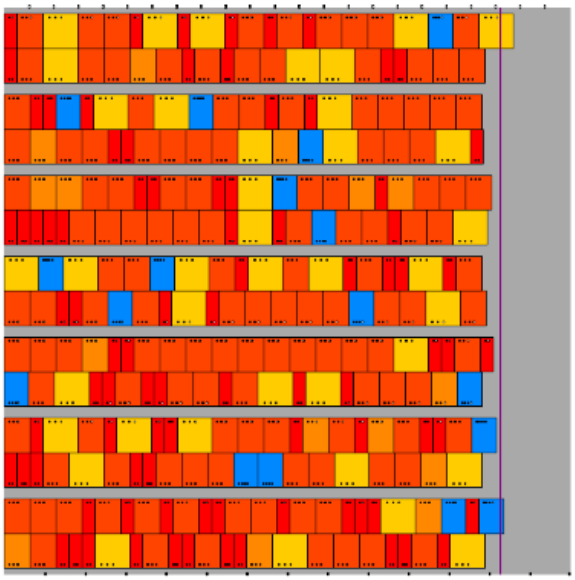
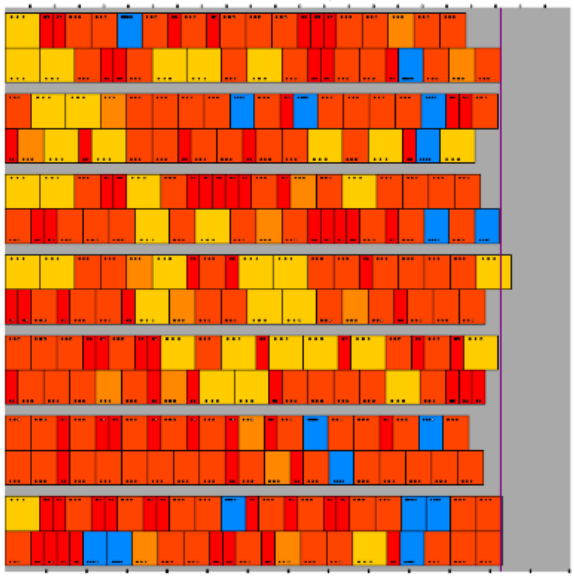


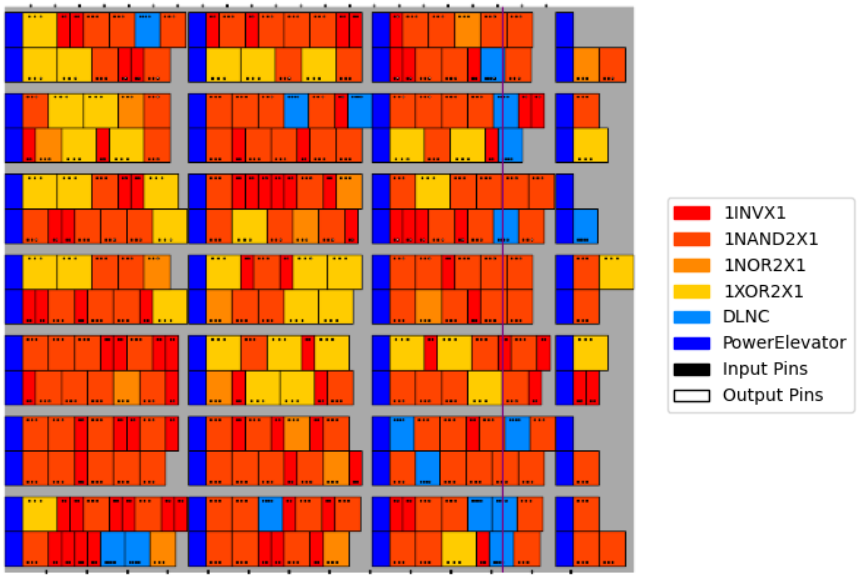
Figure 10 : Simulated annealing of the presented placement.



(a) Random placement.



(b) Placement after simulated annealing.



(c) Final placement with power elevators.

Figure 11 : Stages of the placement process.

2. *Future Work*

In the current implementation, all external output pins are routed to the top of each cell and all external input pins are routed to the bottom of the chip. This somewhat naïve approach can be improved by analyzing the wires in the netlist and placing output and input pins connected by the same wire closer together. Furthermore, the sides of the chip in the routing channels can be used to place external pins. Another important aspect regarding external pin placement is the collaboration with WP5, which must use the external pins of the N^2C^2 blocks to generate the N^2C^2 accelerator.

The quality of the placement generated by simulated annealing is highly dependent on the temperature and cooling rate. Therefore, optimal parameters for the initial temperature and the cooling rate for the simulated annealing must be determined.

III. ROUTING METHOD AND IMPLEMENTATION

1. *Method and prototype*

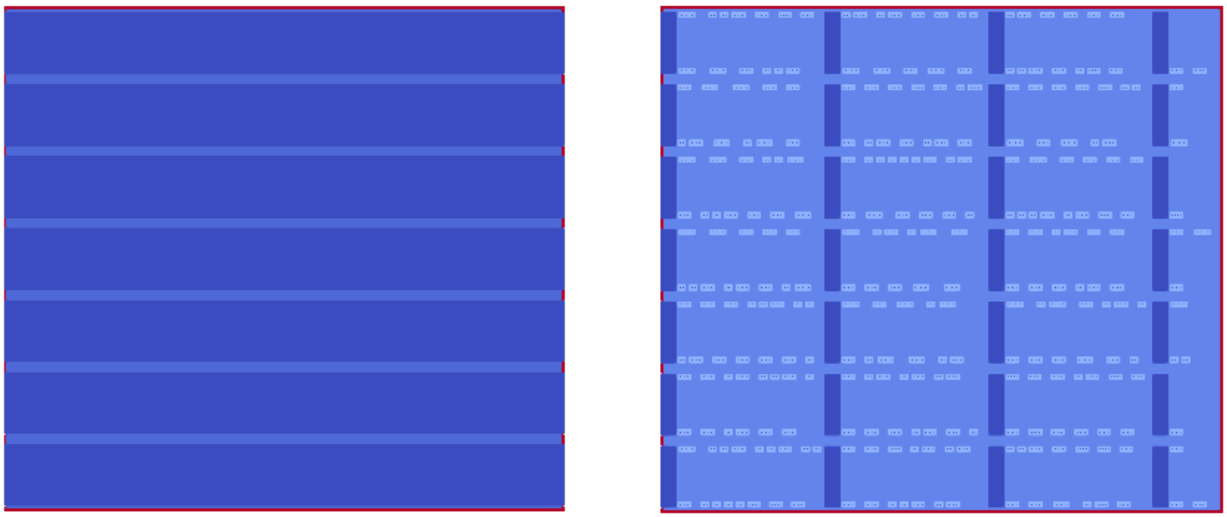
The generation of the physical wires is handled by a *maze routing algorithm* [5]. Maze routing algorithms operate on grids. Therefore, the previously generated layout must be converted into a grid, the so-called *routing grid*.

Each grid cell in the routing grid has a cost (i.e., an integer value) that determines how much it costs to route a physical wire through this grid cell. Furthermore, grid cells can be blocked (i.e., not usable for routing), a grid cell might be blocked if another physical wire is already routed through it, or if a cell or power elevator is occupying the space. The information about whether a cell is blocked or is more expensive to route through is encoded in the so-called *cost function*. This cost function is the primary method that guides the maze routing algorithm (e.g., route physical wires primarily through the routing channel).

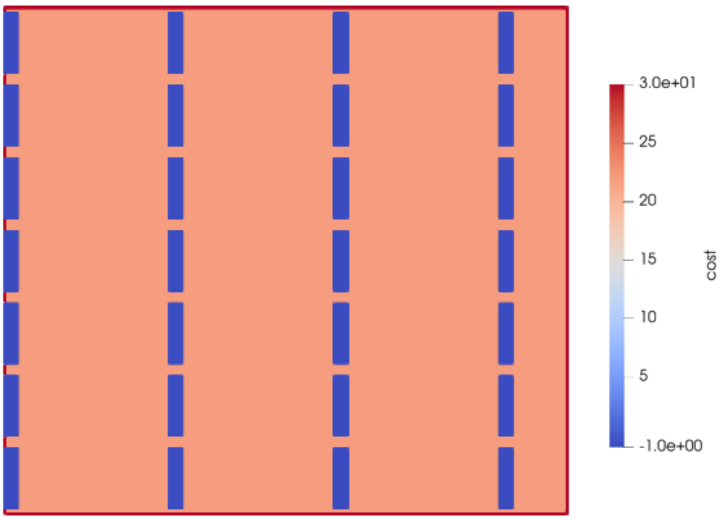
Figure 12 shows the cost function of the metal layers M1, M3 and M5. The cost function is designed in such a way that the cost at the bottom of the routing channel is the lowest and gets higher with each additional metal layer. All cells and power elevators (dark blue) are treated as obstacles. On M3 the pins of the cells can be seen and a small area around the pins with higher cost. This high-cost area is necessary to prevent already created physical wires from blocking access to the pins.

The maze routing algorithm starts at a grid cell in the routing grid called the *source cell* (e.g., a pin) and tries to find a *target cell* (e.g., another pin). This is achieved through a broad first search of the routing grid, starting at the source cell. The search terminates when it finds a target cell. After a target cell has been found a traceback algorithm starts at the target cell, which determines the shortest path back to source cell.

Figure 13 shows the generated physical wires from the top and the bottom of the chip. It can clearly be seen that the wire generation prefers to use the routing channel when a wire must be routed in x-direction. The long physical wires spanning over the entire chip result from the placement of the external pins. These long physical wires can be avoided by optimizing the placement of the external pins as mentioned in the previous section.

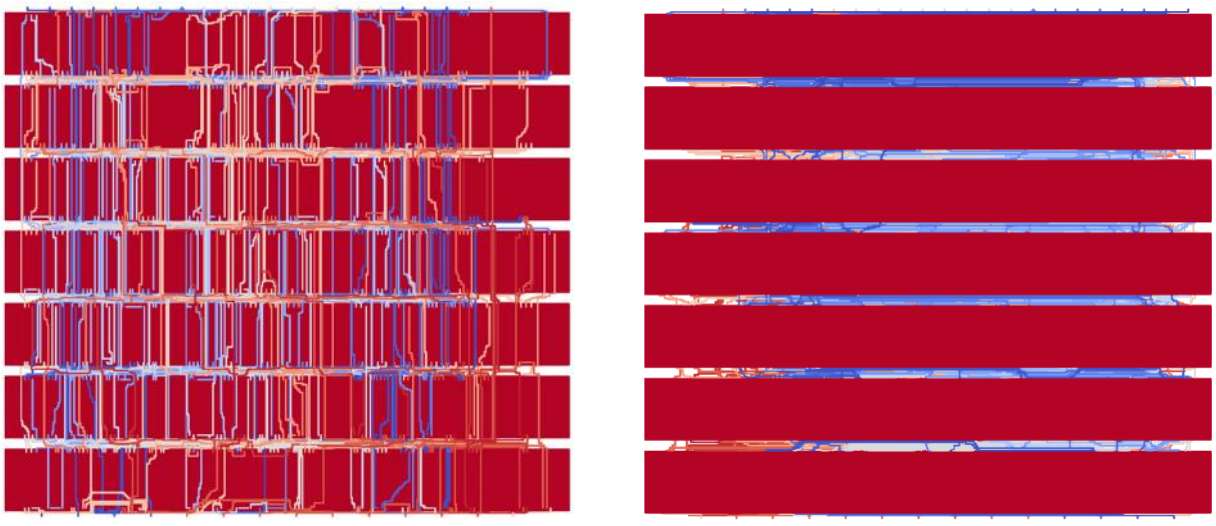


(a) : M1 (b) : M3



(c) : M5

Figure 12 : Cost function of three different metal layers.



(a) : Top (b) : Bottom

Figure 13 : Generated physical wires. Red indicates blocked grid cells. Each other color represents a different physical wire.

A side view of the chip with the generated wires is shown in Figure 14. It can clearly be seen that only the first 4 metal layers are used since most physical wires are routed through the routing channels. The current experiments show that metal layers 5 and 6 are not needed for the Place & Route since M4 is primarily used for routing conflict resolution (i.e., when a physical wire needs to be routed over another physical wire). It should be sufficient to use 4 to 5 metal layers, even for larger chips, since the size of the routing channel can be increased instead of adding additional metal layers.

(a) Side view from the west of the chip.

(b) Side view from the south of the chip.

Figure 14 : Generated physical wires. Red indicates blocked grid cells. Each other color represents a different physical wire.

2. *Future Work*

The maze router requires several upgrades to be used to generate a fabricable layout of the N^2C^2.

- A bend penalty must be implemented to prevent the formation of the step pattern that can be seen on several wires in Figure 13.
- Each metal layer needs a preferred direction for routing, which is important to prevent crosstalk between long parallel running wires.
- The current maze router only uses a single grid, which is sufficient for small netlists like the 4-bit N^2C^2. However, for larger chips the computational costs of routing will become unfeasible when only using a single grid, so, a global router must be developed [6].
- Currently the maze router is implemented in Python, which leads to long maze routing times. Therefore, the maze router must be reimplemented in C++ or Cython to improve performance.

4. Application of P&R tools to 4-bit N^2C^2 using JL1 standard cell library

In this section, we apply the previously described place & route tools to the physical design of the 4-bit N2C2 block as described in section 2. For this, we used the JL1 CStatic standard cell library GDSII files as developed in D4.2 GDSII-ready logic cell and NVM bitcell layout. We first present the technological standard cell data, then describe the successful generation of the physical results and the full layout of the 4-bit N2C2. Finally, we identify perspectives for future work.

I. JL1 STANDARD CELL LIBRARY PHYSICAL DATA

In order to move towards the application of the place & route tools to the synthesis of functional blocks based on real technology, we generated a .lef library file containing the physical data for CStatic logic cells implemented in JL1 technology. We generated a preliminary physical layout for the cells in the standard cell library as described above, and detailed in deliverable D4.2 GDSII-ready logic cell and NVM bitcell layout. These layouts were generated in GDSII file format and based on design rules extended from ASAP 7nm FinFET DRM to incorporate the vertical nanowires technology. This allowed us to obtain the height and width of the cells and thus their area. Table 4 summarizes this information for each cell. All the cells have the same height to enable conventional row-placement, as indicated previously; however, their width varies according to the complexity of the cell as well as the number of nanowires used per cell.

Table 4 JL1 CStatic standard logic cell dimensions

Gate	Width (μm)	Height (μm)	Area (μm^2)
1INVX1	0.166	0.688	0.114
2INVX1	0.577	0.688	0.397
3INVX1	0.993	0.688	0.683
4INVX1	1.41	0.688	0.970
1NAND2X1	0.449	0.688	0.309
2NAND2X1	2.125	0.688	1.462
3NAND2X1	3.82	0.688	2.628
1NOR2X1	0.572	0.688	0.394
2NOR2X1	2.104	0.688	1.448
3NOR2X1	3.945	0.688	2.714
1XOR2X1	1.309	0.688	0.901
2XOR2X1	5.529	0.688	3.804
3XOR2X1	9.695	0.688	6.67016
DFFSR	6.586	0.688	4.531

II. N2C2_JL1 PHYSICAL SYNTHESIS RESULTS

Current development efforts for physical design tools were focused on developing a placement and a physical routing algorithm for the VNWFET technology. The physical wires generated by the maze router are individually accessible and are still connected to their logical representation in the netlist. Therefore, the properties of the wires (e.g., wire length) can be determined and used as inputs for resistance calculation algorithms (e.g., analytical solvers or field solvers). The implementation of such resistance calculation algorithms will commence when the additional features described in Section 3.III have been implemented.

The results of the tools as applied to the 4-bit N^2C^2 block using the .lef file describing the JL1 library of CStatic logic cells are shown in the figures below. This experiment, with 230 cells and 315 wires, took 3.5 days to run on the GTS cluster and will be a focus for optimization as indicated in the previous section.

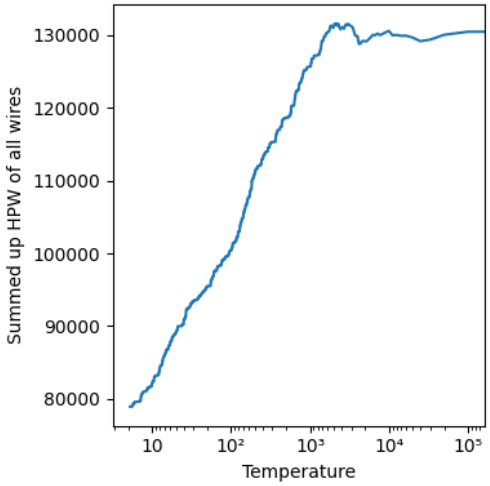
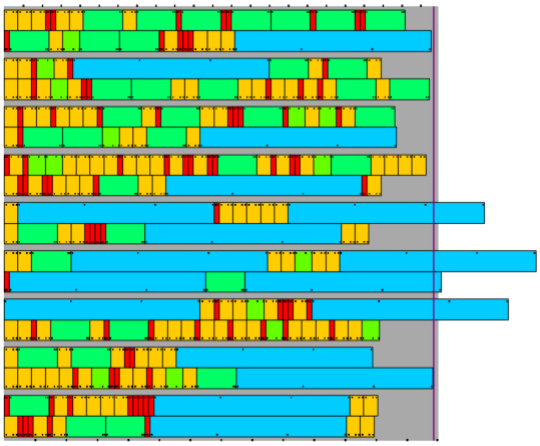
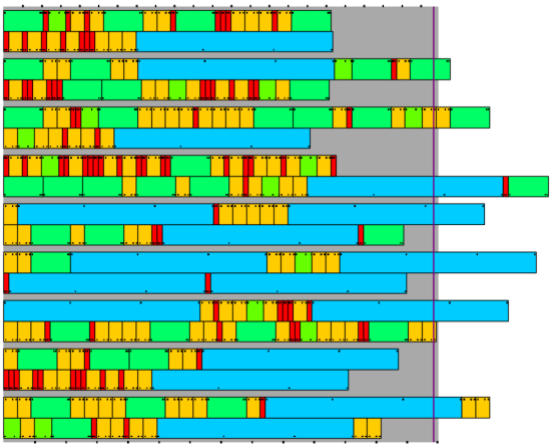


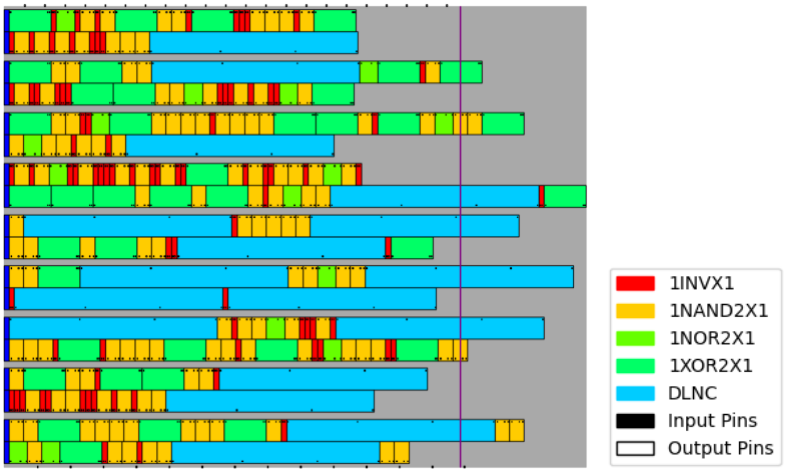
Figure 15 : Simulated annealing of the presented placement.



(a) Random placement.



(b) Placement after simulated annealing.



(c) Final placement with power elevators.

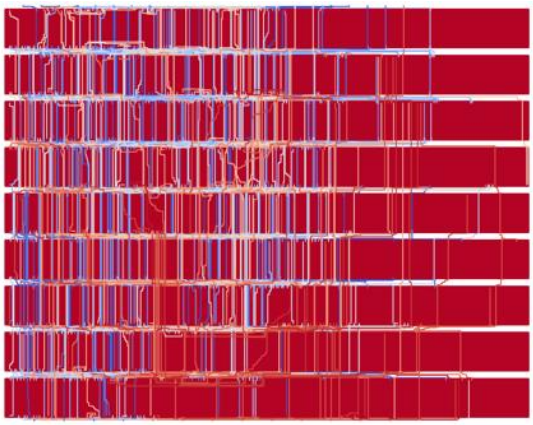
Figure 16 : Stages of the placement process.

(a) : M1

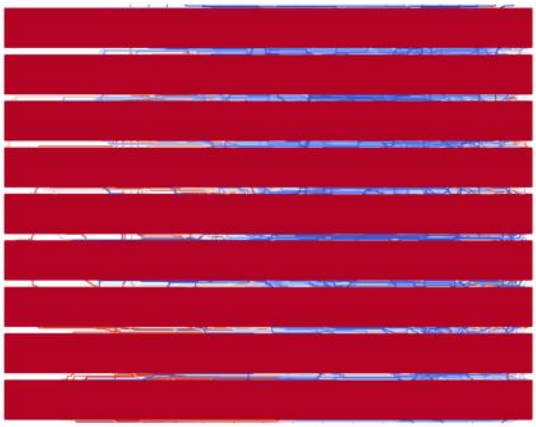
(b) : M3

(c) : M5

Figure 17 : Cost function of three different metal layers.



(a) : Top



(b) : Bottom

Figure 18 : Generated physical wires. Red indicates blocked grid cells. Each other color represents a different physical wire.

(a) Side view from the west of the chip.

(b) Side view from the south of the chip.

Figure 19 : Generated physical wires. Red indicates blocked grid cells. Each other color represents a different physical wire.

III. DISCUSSION ON 4-BIT N2C2 LAYOUT

While this first generated layout of the 4-bit N2C2 block using JL1 standard cells was successful and "closes" the circuit-scale part of the DTCO loop, multiple improvements are possible for future work:

- Large variations in cell aspect ratios are clearly visible in Figure 16, particularly for the D-type flip-flop DLNC. Further work is required on the actual layouts of the CStatic JL1 library, to reduce whitespace in the logic cells and reduce footprint.
- The use of JL2 technology is expected also to significantly improve cell footprint, in particular for the XOR functions.
- The generation of optimized N2C2 blocks will enable the extraction of routing parasitics for full N2C2 energy-delay analysis.
- Tool optimization (to improve runtime) is necessary to enable scaling to larger size N2C2. Due to the greater potential for place & route (larger design space and options to reduce sensitivity to specific cell sizes), it is anticipated that this will also improve overall layout homogeneity.
- 3D layout and cell stacking / routing in the vertical dimension will also be explored.

5. Conclusion

In this deliverable, we described the design of a scaled-down (4-bit) Neural Network Compute Cube (N^2C^2). This is a concrete use-case using multiple strands from FVLLMONTI WP3 and WP4 tasks:

- logic synthesis on a "toy" 4-bit N^2C^2 circuit using the the library of JL1 CStatic logic cells described in D4.1 Library of optimized VNWFET-based logic cells with the extracted timing and power characteristics
- development of specific place and route methods and tools for the physical design of N^2C^2 using the intra-N^2C^2 interconnect framework defined in D3.3 3D-place-and-route prototype – V1 and automated layout and GDSII generated JL1 CStatic logic cells described in D4.2 GDSII-ready logic cell and NVM bitcell layout

This is a necessary step towards a) establishing a full toolchain from device to complex logic block and b) enabling the gathering of first insights into the actual design, performance and cost of the N^2C^2 block based on vertical technology.

We first reviewed the design of a scaled-down N^2C^2. We covered the overall functional structure, and discussed the scalability of the N^2C^2 block. The view in this deliverable was to focus on a scaled-down (4-bit) version of the block in order to test the end-to-end synthesis methodology and toolflow, as well as to gather first insights as to key performance indicators for the approach. A description of the data and logic cell library file then follows, enabling subsequent synthesis and testing for several cases of interest.

We then described in more detail the physical design method, which is carried out in two steps:

1. Generation of the placement of the cells on the chip surface. In our approach, this placement uses heuristics to estimate the expected length of the wires in the netlist, which is used to place the cells such that the expected overall wire length is minimized.
2. Generation of the physical connections (i.e., physical wires) between the cells. Here, we use a maze routing algorithm which is guided by a cost function that generates physical wires for all wires in the netlist.

To generate the physical layout of the N^2C^2, a prototype for a Place & Route tool has been implemented in Python. All results presented were generated with this prototype.

Finally, we then described the results of the tools as applied to the 4-bit N^2C^2 block using the .lef file describing the JL1 library of CStatic logic. This experiment, with 230 cells and 315 wires, took 3.5 days to run on the GTS cluster and will be a focus for optimization as indicated in the previous section.

While this first generated layout of the 4-bit N2C2 block using JL1 standard cells was successful and "closes" the circuit-scale part of the DTCO loop, multiple improvements are possible for future work, from reworking GDSII files for JL1 and JL2 technology, to optimizing the tool performance for scalability, as well as incorporating 3D stacking and routing.

6. References

[1] A. Poittevin, I. O'Connor, G. Larrieu, C. Maneux, C. Mukherjee, "Compact modeling of 3D vertical junctionless gate-all-around silicon nanowire transistors towards 3d logic design", "Solid-State Electronics" vol. 183, p. 108125, 2021

[2] D. L. et al., "Many-Tier Vertical Gate-All-Around Nanowire FET Standard Cell Synthesis for Advanced Technology Nodes", IEEE Journal on Exploratory Solid-State Computational Devices and Circuits, 2021

[3] Garey, M. R., & Johnson, D. S. (1990). Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & co.

[4] Wong, D. F., Leong, H. W., & Liu, C. L. (2012). Simulated Annealing for VLSI Design. Springer Science & Business Media

[5] Lee, C. Y. (1961). An Algorithm for Path Connections and Its Applications. IRE Transactions on Electronic Computers, 346-365

[6] Hu, J., & Sapatnekar, S. S. (2001). A Survey on Multi-Net Global Routing for Integrated Circuits. Integration, 1-49