A Study on Advanced Ising Machine Designs: BLIM, King's Graph Mapping, and OIM Chips

Jui-Hsin Hung
Jaijeet Roychowdhury

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2024-95
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2024/EECS-2024-95.html

May 10, 2024

A Study on Advanced Ising Machine Designs: BLIM, King's Graph Mapping, and OIM Chips

by Jui-Hsin Hung

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Jaijeet Roychowdhury
Research Advisor

May 9, 2024

(Date)

* * * * * * *

Professor Jun-Chau Chien
Second Reader

May 9, 2024

(Date)

Abstract

This report investigates three pivotal areas of Ising machine technology: the Bistable Latch Ising Machine (BLIM), the mapping of Ising problems onto King's graph, and the design of Oscillator Ising Machine (OIM) chips. Our study of BLIM reveals its potential in solving small-scale Ising problems, though further research is needed to enhance its scalability. The exploration of Ising problem mapping onto King's graph examines the effectiveness of minor embeddings and coupling coefficient assignments. Although the approach is methodically sound, it has shown limited success in achieving optimal Ising Hamiltonians and raises concerns about its practicality due to increased hardware demands. Additionally, we detail the design processes for two types of OIM chips, highlighting the architectural differences and operational challenges encountered during tapeouts. Overall, while each area demonstrates significant promise, ongoing development and validation are critical to fully realize their potential in practical applications.

Contents

1 Introduction

An Ising machine is a piece of electronic hardware designed to find the spin configuration with minimum Ising Hamiltonian, i.e., to solve the Ising problem. The Ising Hamiltonian is defined as

$$H(\vec{s}) = -\frac{1}{2}\sum_{i,j=1}^{N} J_{ij}s_i s_j, \tag{1.1}$$

where $\vec{s}$ are the spins, i.e., $\vec{s} = [s_1, s_2, \ldots, s_N]^T$, with $s_i \in \{+1, -1\}$; J_{ij} are the coupling coefficients, with $J_{ij} = J_{ji}$ and $J_{ii} = 0$.

This report explores two types of Ising machines: the Bistable Latch Ising Machine (BLIM) and the Oscillator Ising Machine (OIM), as proposed in [1] and [2] respectively. In BLIM, a spin is implemented using a latch (bistable element), while OIM employs a nonlinear oscillator. The coupling coefficients are implemented using resistors in both machines. Through connecting these latches using coupling resistors, a system that functions as an Ising machine is formed.

The remainder of the report is structured as follows: Section 2 outlines the research activities conducted on BLIM, including theoretical developments, simulations, and experimental measurements. Section 3 explores the process of mapping Ising problems onto King's graph, providing an example of how Ising problems could potentially be implemented on hardware platforms. Section 4 elaborates on my contributions to the design process of two OIM chips, detailing the roles undertaken, challenges encountered, and solutions implemented. Finally, Section 5 offers the conclusion, summarizing the key findings and contributions of this study.

2 BLIM

At the start of this section, we introduce a simplified model of the Bistable Latch Ising Machine (BLIM) in Section 2.1, where we outline the theoretical principles supporting its operation. Following this, Section 2.2 explores simulations to further clarify the workings of BLIM. Finally, Section 2.3 presents practical measurements performed on a breadboard, showcasing BLIM's efficacy in addressing small-scale Ising problems. Through this structured approach, we aim to highlight BLIM's potential in computational problem-solving.

2.1 Theory

In a Bistable Latch Ising Machine (BLIM), individual spins of the Ising model are realized through latches, with each latch corresponding to a single spin. The resistive connections between these latches represents the coupling coefficients, thereby establishing the network of spins in a physical form.

In this project, a latch is built using two back-to-back inverters. Following [1], a simplified model of the latch includes one inverter capacitor, as shown in Fig. 1. Note that a typical inverter's voltage characteristic curve is loosely similar to $\tanh(\cdot)$, hence we use the simple analytical model

$$v_o = \tanh(-kv_i), \tag{2.1}$$

where the gain parameter $k > 0$ can be implemented approximately by changing V_{DD}. We define a parameter

$$K_s = V_{DD} - V_{SS}, \tag{2.2}$$

where a high K_s corresponds to a steep slope in tanh, i.e., high k; while a low K_s corresponds to a flat slope in tanh, i.e., low k. This K_s would be used for parameter cycling, which is described in more detail in the next section.

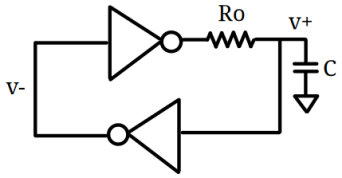


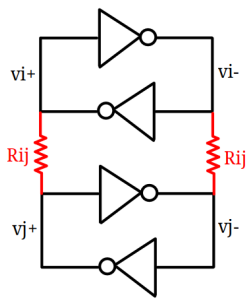
Figure 1: Simplified model of a latch built with 2 inverters, including an output resistance and a load capacitor at one of the output of the inverters, which is v_+ in the figure.

Each coupling, implemented between 2 latches, uses a pair of coupling resistors. Depending on the sign of J_{ij}, there are two ways to connect the coupling resistors, as shown in Fig. 2. For positive J_{ij}, one of the coupling resistor connects v_{i+} and v_{j+}; the other one connects v_{i-} and v_{j-}. For negative J_{ij}, one of the coupling resistor connects v_{i+} and v_{j-}; the other one connects v_{i-} and v_{j+}. We define $v_i = v_{i+}$ in the following content. The resistance of a coupling resistor R_{ij} corresponds to the value of a coupling coefficient, i.e.,

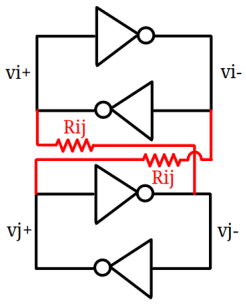
$$|Jij| = \frac{K_c}{R_{ij}}. \tag{2.3}$$

Expressing $J_{ij} = \text{sgn}(J_{ij})|J_{ij}|$ and using the definition in (2.3), we rewrite the term $\frac{1}{R_{ij}}$ as

$$\frac{1}{R_{ij}} = \text{sgn}(J_{ij})\frac{J_{ij}}{K_c} = \begin{cases} \frac{J_{ij}}{K_c} & \text{, if } J_{ij} > 0, \\ -\frac{J_{ij}}{K_c} & \text{, if } J_{ij} < 0. \end{cases} \tag{2.4}$$



(a) Implementing positive J_{ij}.



(b) Implementing negative J_{ij}.

Figure 2: BLIM models that implement positive and negative coupling.

For a BLIM with N coupling latches, a system of N differential equations results ([1], Appendix A):

$$\frac{dv_i(t)}{dt} = f(v_i; k) - \sum_{j=1}^{N} J_{ij}\ g(v_i, v_j, \text{sgn}(J_{ij}); k),\ \ i = 1, \ldots, N, \tag{2.5}$$

where

$$f(v_i;k) = \frac{G}{C}(\tanh(k\tanh(kv_i)) - v_i), \tag{2.6a}$$

$$g(v_i, v_j, \mathrm{sgn}(J_{ij});k) = \begin{cases} \frac{1}{K_c C}(v_i - v_j) & , \text{if } J_{ij} > 0, \\ \frac{-1}{K_c C}(v_i - \tanh(-kv_j)) & , \text{if } J_{ij} < 0. \end{cases} \tag{2.6b}$$

$$G = \frac{1}{R_o}, \tag{2.6c}$$

$$\text{and } |Jij| = \frac{K_c}{R_{ij}}. \tag{2.6d}$$

The Lyapunov function for this system is [1]

$$L_N(\vec{v};k) = -\left(\sum_{i=1}^{N} z(v_i;k) - \sum_{i,j=1}^{N} J_{ij}h(v_i,v_j;k)\right). \tag{2.7}$$

where

$$z(v;k) = \int_0^v f(x;k)\,dx\ , \tag{2.8a}$$

$$\text{and } h(v_i, v_j, \mathrm{sgn}(J_{ij});k) = \begin{cases} \frac{1}{2K_c C}[\frac{(v_i - v_j)^2}{2} - 1] & , \text{if } J_{ij} > 0, \\ \frac{-1}{2K_c C}[\frac{(v_i + v_j)^2}{2} - 1] & , \text{if } J_{ij} < 0. \end{cases} \tag{2.8b}$$

According to [1], the Lyapunov function, (A.3), equals a scaled/shifted Ising Hamiltonian when $k > 0$ is large that the spin voltages are at the bistable values v_+ and v_-. The corresponding spin is defined as

$$s_i = \begin{cases} 1 & , \text{if } v_i = v_+, \\ -1 & , \text{if } v_i = v_-. \end{cases} \tag{2.9}$$

As the BLIM model with positive coupling, i.e., $J_{ij} > 0$, has been proven to function as an Ising machine in [1], the derivation of negative coupling, i.e., $J_{ij} < 0$, is provided in Appendix A.

2.2 Simulation

Transient simulations of 2/3/4/5/10/20-spin BLIM were conducted using SPECTRE. The definition of problems we simulated on is attached in Appendix B. Moreover, bruteforce simulations were executed on C to determine the minimum Ising Hamiltonian of each problem. These simulations involve computing the Hamiltonian for all possible spin configurations, providing a comprehensive distribution of Hamiltonian values.

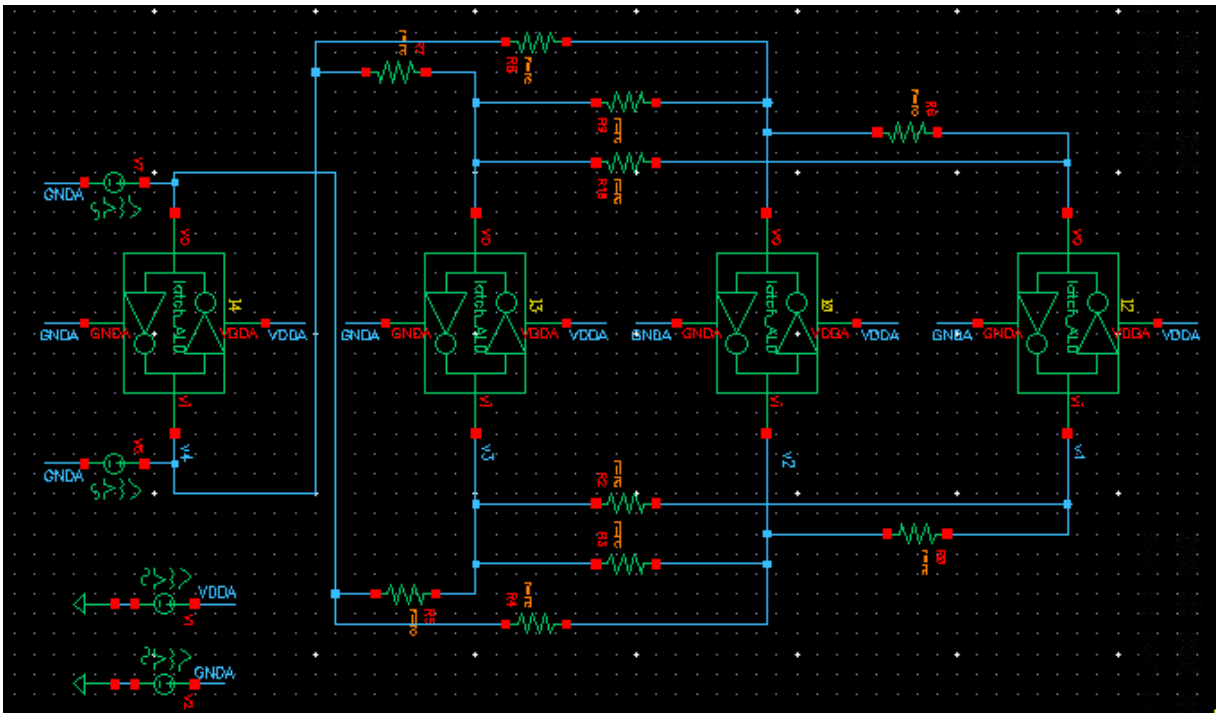
In the transient simulations, the latches are implemented using ALD1106 (NMOS) and ALD1107 (PMOS) as SPICE level 2 MOSFET models. Also, the latches are coupled through resistors representing J_{ij} in Ising problems, as defined in (2.3). A schematic of a `size-4` BLIM is shown in Fig. 3 as an example.

Additionally, parameter cycling was introduced as Ising problems need parameter cycling to achieve lower Hamiltonian values [1] [1]. In this project, we use a pulse waveform for V_{DD}

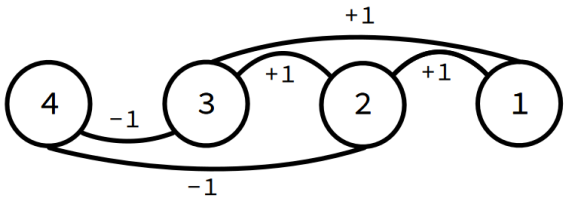
[1] It is not entirely clear how parameter cycling influences the Hamiltonian and the voltage of the latch in BLIM. According to [1], it can be seen that successive parameter cyclings on Ks reduces the Hamiltonian, particularly when Ks is low, i.e., the latch system is in analog mode. Conversely, when Ks is set high, the system is binarized again. However, the mechanism behind this feature is not well understood yet.

to cycle the parameter K_s. When V_{DD} is high, the voltages of the latches will be binarized into $v_i \in \{v_+, v_-\}$; when V_{DD} is low, the binarization is eliminated and the Hamiltonian is reduced. Examples of transient simulation waveforms are shown in Fig. 4.

Following [1], we map $v_i > 0$ to $s_i = +1$ and $v_i < 0$ to $s_i = -1$. The final Ising Hamiltonian results of transient simulation of 2/3/4/5/10/20-spin BLIM were compared with bruteforce solution, as shown in Table 1. With the number of spin and the number of edge increase, the value of K_c also needs to increase to achieve a better Hamiltonian. Note that BLIM finds the minimum Hamiltonian for most of these small problems. Among them, there are 3 problems, `J20_11`, `J20_14`, and `J20_15`, that the result minimum Hamiltonian is much worse even after many trials of adjusting parameter values.

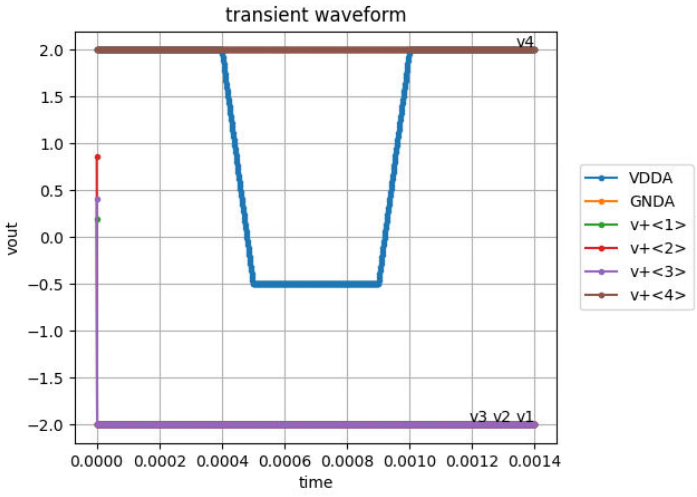


(a) Schematic of `size4_1` problem implemented in BLIM. Each latch (green) implements a spin s_i in (1.1); each resistor implements a coupling coefficient J_{ij}.

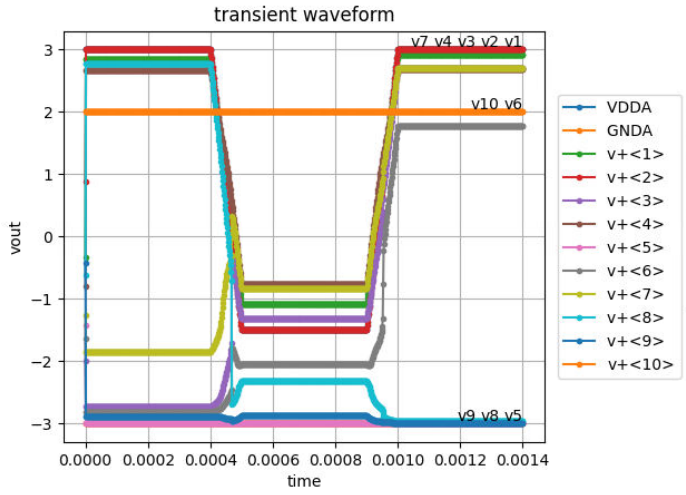


(b) Graph of the `size4_1` problem.

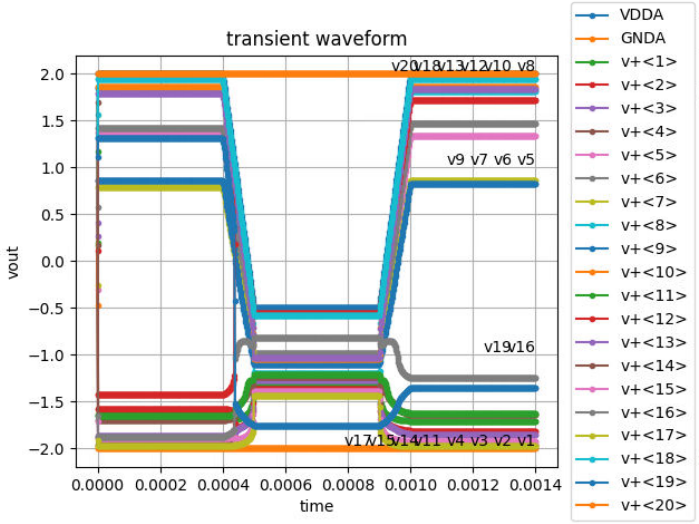
Figure 3: Schematic and graph of the `size4_1` problem.



(a) Simulation waveform of the problem `size4_1`.



(b) Simulation waveform of the problem `J10_12`.



(c) Simulation waveform of the problem `J20_15`.

Figure 4: Simulation waveforms of BLIM with a $1G$ Hz pulse V_{DD}. The time step is $1.4\mu s$ and the stop time is $1.4ms$. The parameter cycling of K_s happens at $t = 0.4m \sim 1ms$, when V_{DD} is lower. The voltage v_N is fixed at v_+.

Table 1: Simulation results and their corresponding parameter settings. The count is the number of times that the minimum simulated Ising Hamiltonian was reached in 200 runs. The period of pulse V_{DD} is $1ms$. The time step is $1.4\mu s$ and the time stop is $1.4ms$.

problem name	simulation $\min(H(\vec{s}))$ {value, count}	bruteforce $\min(H(\vec{s}))$	K_s {high, low}	K_c	$V_{DD,high}, V_{DD,low}, V_{SS}, V_n$
size2_1	-1, 200	-1	4, 1.5	2k	2, -0.5, -2, 2
size2_2	-1, 200	-1	4, 1.5	2k	2, -0.5, -2, 2
size3_1	-3, 200	-3	4, 1.5	2k	2, -0.5, -2, 2
size3_2	-3, 200	-3	4, 1.5	2k	2, -0.5, -2, 2
size3_3	-1, 200	-1	4, 1.5	2k	2, -0.5, -2, 2
size4_1	-5, 200	-5	4, 1.5	2k	2, -0.5, -2, 2
size4_2	-5, 200	-5	4, 1.5	2k	2, -0.5, -2, 2
size5_1	-6, 200	-6	4, 1.5	2k	2, -0.5, -2, 2
J10_01	-9, 200	-9	4, 1.5	2k	2, -0.5, -2, 2
J10_02	-9, 200	-9	4, 1.5	2k	2, -0.5, -2, 2
J10_03	-9, 200	-9	4, 1.5	2k	2, -0.5, -2, 2
J10_04	-9, 200	-9	4, 1.5	2k	2, -0.5, -2, 2
J10_05	-9, 200	-9	4, 1.5	2k	2, -0.5, -2, 2
J10_06	-9, 200	-9	4, 1.5	2k	2, -0.5, -2, 2
J10_07	-7, 200	-7	4, 1.5	2k	2, -0.5, -2, 2
J10_08	-9, 200	-9	4, 1.5	2k	2, -0.5, -2, 2
J10_09	-9, 200	-9	4, 1.5	2k	2, -0.5, -2, 2
J10_10	-7, 200	-7	4, 1.5	2k	2, -0.5, -2, 2
J10_11	-532, 200	-532	4, 1.5	2k	2, -0.5, -2, 2
J10_12	-424, 200	-424	6, 1.5	54k	3, -1.5, -3, 2
J10_13	-604, 200	-604	4, 1.5	2k	2, -0.5, -2, 2
J10_14	-476, 200	-467	4, 1.5	2k	2, -0.5, -2, 2
J10_15	-701, 200	-701	4, 1.5	2k	2, -0.5, -2, 2
J20_01	-38, 200	-38	4, 1.5	4k	2, -0.5, -2, 2
J20_02	-38, 200	-38	4, 1.5	4k	2, -0.5, -2, 2
J20_03	-38, 200	-38	4, 1.5	4k	2, -0.5, -2, 2
J20_04	-38, 200	-38	4, 1.5	2k	2, -0.5, -2, 2
J20_05	-38, 200	-38	4, 1.5	4k	2, -0.5, -2, 2
J20_06	-24, 200	-24	4, 1.5	8k	2, -0.5, -2, 2
J20_07	-22, 200	-22	4, 1.5	10k	2, -0.5, -2, 2
J20_08	-28, 200	-28	6, 1.5	2k	3, -0.5, -3, 2
J20_09	-24, 200	-24	4, 1.5	4k	2, -0.5, -2, 2
J20_10	-26, 200	-26	4, 1.5	4k	2, -0.5, -2, 2
J20_11	-1412, 108	-1432	6, 1.5	275k	3, -1.5, -3, 2
J20_12	-1568, 200	-1568	4, 1.5	10k	2, -0.5, -2, 2
J20_13	-1226, 200	-1226	4, 1.5	20k	2, -0.5, -2, 2
J20_14	-1647, 200	-1701	6, 1.5	120k	3, -1.5, -3, 2
J20_15	-1421, 200	-1427	6, 1	150k	3, -2, -3, 2

2.3 Measurement

A 4-spin BLIM was implemented on a breadboard, as shown in Fig. 5. Note that it can also be configured to implement fewer-spin problems by adjusting the configuration of coupling resistors. The latches were implemented with ALD1106 (NMOS) and ALD1107 (PMOS) chips. Considering that all the values of $|J_{ij}|$ in our 4-spin problems are 1, the resistance of the coupling resistors is implemented using resistors with fixed resistance 150Ω, i.e., we choose $K_c = 150$. Voltage waveforms were measured using a four-channel oscilloscope; a photo of a measurement result is shown in Fig. 6.

To translate from voltages to spin voltages [1], we map $v_i > \frac{VDD}{2}$ to $s_i = +1$ and $v_i < \frac{VDD}{2}$ to $s_i = -1$, then obtain the Ising Hamiltonian using (1.1). The measurement results are compared with simulation and bruteforce solution, as shown in Table 2. Though we haven't implemented parameter cycling on V_{DD} as in simulation, we still got the minimum Ising Hamiltonian for the small size problems. To verify, we performed 10 measurements on problems `size4_1` and `size4_2` respectively, and all the minimum Ising Hamiltonians were obtained immediately without parameter cycling.

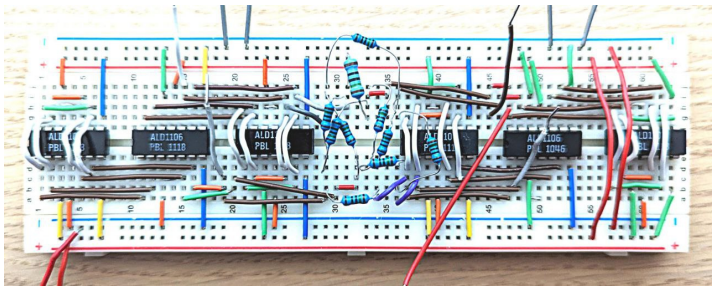


Figure 5: Photo of a size-4 BLIM implemented on breadboard. The 1st, 3rd, 4th,and 6th chips (black) from left are ALD1107 (4 PMOS/pkg); the 2nd and 5th chips (black) are ALD1106 (4 NMOS/pkg). An inverter is built with 2 PMOS and 1 NMOS devices. The couplings between the spins are implemented by resistors (blue).

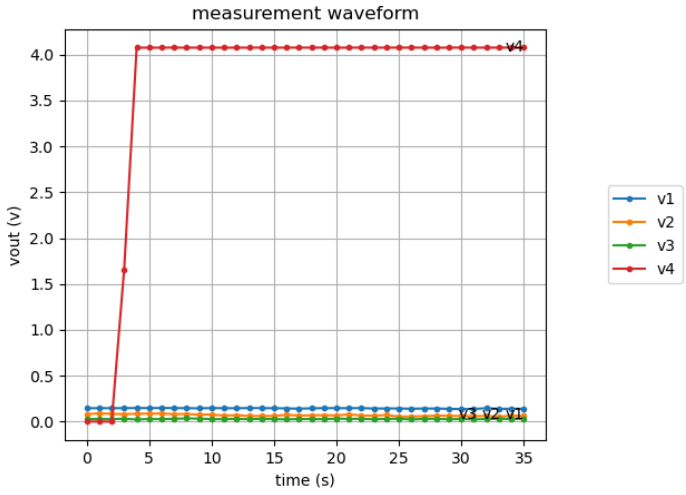


Figure 6: Measured waveform of the `size4_1` problem. V_{DD} is turned on at $t = 3$s and fixed at 4V. The final spins voltages $\{v_1, v_2, v_3, v_4\}$ are $\{0.14, 0.06, 0.03, 4.08\}$V respectively, and the corresponding spins $\{s_1, s_2, s_3, s_4\}$ are $\{-1, -1, -1, +1\}$, achieving the minimum Ising Hamiltonian of problem `size4_1`.

Table 2: Measurement result. V_{DD} is fixed at 4V and the coupling resistance R_c is 150Ω. The minimum Hamiltonian was achieved in both measurement and simulation.

problem name	measurement Hamiltonian	simulation Hamiltonian	bruteforce min(Hamiltonian)	K_s	K_c
size2_1	-1	-1	-1	4	150
size2_2	-1	-1	-1	4	150
size3_1	-3	-3	-3	4	150
size3_2	-3	-3	-3	4	150
size3_3	-1	-1	-1	4	150
size4_1	-5	-5	-5	4	150
size4_2	-5	-5	-5	4	150

2.4 Summary

Through theory derivation, simulation, and measurement, we have demonstrated the potential of BLIM. However, there are still avenues for further exploration to solidify its capabilities. In theory, the derivation of the distinction between single-sided and double-sided coupling remains unresolved. Regarding simulation, there is potential in experimenting with different input sources, such as using pulse voltage on the fixed node, applying pulses to VDD and VSS, or employing ladder VDD to assess the impact of varying Kc on results. In terms of measurement, constructing larger problem sets, like the J10 problem, on a breadboard could offer practical validation. These avenues for future investigation aim to provide additional evidence of BLIM's effectiveness.

3 King's Graph Mapping

A King's graph represents all the possible moves of a king on the chessboard. In this graph, each vertex corresponds to a square on the chessboard, and the edges represent the king's possible moves between these squares in a single step. An example of 4×4 King's graph, denoted as $KG_{4,4}$, is shown in Fig. 7.

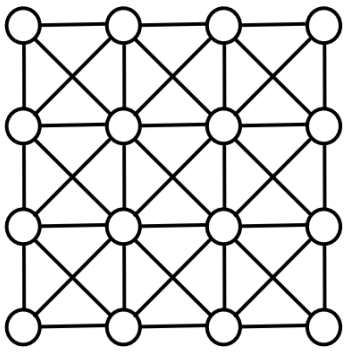


Figure 7: Graph of $KG_{4,4}$.

In this section, we explore the process of mapping an arbitrary input graph, also considered as an Ising problem, onto the King's graph. This mapping process involves two steps: first, the identification of a minor embedding graph, and second, the precise assignment of weight coefficients to establish the necessary connections.

3.1 Finding a Minor Embedding

To translate an input graph $I = (V(I), E(I))$, where $V(I)$ is a set of vertices with spin indices $i = 1, ..., N$ and $E(I)$ is a set of edges (i, j) corresponding to edges with non-zero weights J_{ij}, into a King's graph $H = (V(H), E(H))$, the concept of minor embedding must be introduced. This process involves mapping each vertex $i \in V(I)$ to one or more vertices in the King's graph, where those vertices forms a subset of vertex $\phi(i) \in V(H)$, called super vertex placement. A successful mapping is achieved when all edge connectivities in $E(I)$ are preserved in $E(H)$, making ϕ a minor embedding of the input graph.

Two mapping techniques have been implemented in our content: the best known complete graph embedding method [5], and the probabilistic-swap-shift-annealing (PSSA) embedding heuristic [4]. These techniques will be discussed further in the subsequent sections.

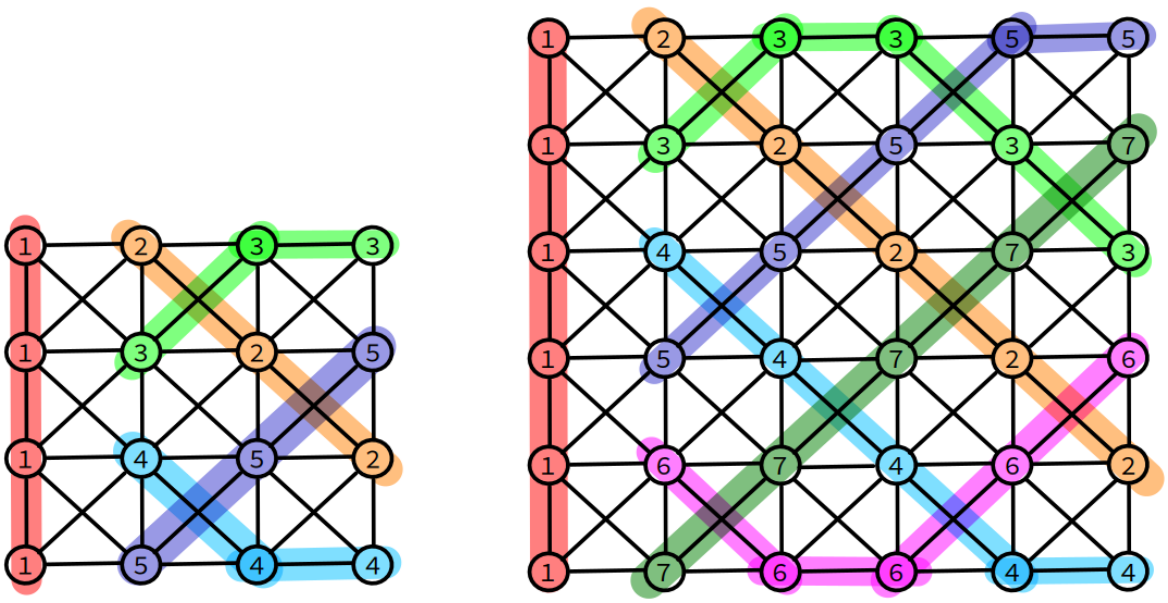
3.1.1 Best Known Complete Graph Embedding

The authors of [4] defined the graph pattern in [5] as the best known complete graph embedding. This pattern is formed on a King's Graph, $KG_{L,L}$, with a width of $L = N - 1$. It serves as a minor embedding of an N-spin complete graph, K_N and is considered the best due to the fact that it is impossible to map a graph K_N onto a King's Graph $KG_{L,L}$ where $L <= N - 2$.

The mapping process can be broken down into the following steps:

1. Mapping vertex 1 to $\phi(1)$:
    - Vertex 1 of the input graph is assigned to all vertices in the first column of the King's Graph.
2. Mapping Vertices i to $\phi(i)$, $i = 2, \ldots, N$:
    - Even-indexed vertices:
        - Start their path from row $i - 1$ in the second column.
        - Move diagonally down-right until reaching the bottom boundary of the grid.
        - Then, move one space to the right and continue diagonally up-right toward the last column.
    - Odd-indexed vertices:
        - Start their path from row $i - 1$ in the second column.
        - Move diagonally up-right until reaching the top boundary of the grid.
        - Then, move one space to the right and continue diagonally down-right toward the last column.

Following these rules ensures that each $\phi(i)$ maintains at least one edge connecting to every $\phi(j)$, where $j = 1, \ldots, N$, $j \neq i$, on the King's Graph. This guarantees that the connectivity of the input graph are always representable within the $L \times L$ grid of the King's Graph. Examples illustrating the best known complete graph embedding are provided in Fig. 8. Note that any N-spin graph can be mapped onto a $KG_{L,L}$ using this method, with non-existing edges in the input graph assigned zeros weights on the output graph.



(a) $KG_{4,4}$, the best known complete graph embedding of 5-spin complete graph, K_5.

(b) $KG_{6,6}$, the best known complete graph embedding of 7-spin complete graph, K_7.

Figure 8: Examples of best known complete graph embedding. The number on each vertex represents the index of the vertex in the input graph that it maps to. In other words, the vertices with the same number i are in the same subset of vertices $\phi(i)$ in King's graph.

3.1.2 PSSA Graph Embedding

The probabilistic-swap-shift-annealing (PSSA) embedding heuristic, introduced by Sugie et al. [4], is an algorithm specifically designed to map an N-spin sparse graph onto a King's graph $KG_{L,L}$, where $L < N - 1$. This method employs the principles of simulation annealing to explore the minor embedding of the graph.

The mapping process can be divided into an initial stage followed by an iterative stage. In the initial stage, super vertex placement is initialized based on a guiding pattern derived from the best-known complete graph. Subsequently, the iterative stage executes, repeating until either the minor embedding is successfully found or a predefined maximum number of iterations is reached.

During each iterative step, the algorithm performs a movement on vertices, either a swap or a shift, determined by sampled results. These movements involve the selection of random spin indices and the formation of a new super vertex placement, referred to as a proposal. After each movement, the algorithm evaluates how many vertices in $E(I)$ have been successfully formed in $E(H_{\text{proposal}})$. The minor embedding is considered established when all the vertices are formed and all the edges of the input graph are properly embedded into the King's graph.

3.2 Mapping the Weight Coefficients

A proper mapping of weight coefficient is important to ensure that the output graph maintains a shifted version of the Ising Hamiltonian of the input graph. A mapping method outlined in [6] employs a simple upper bound for coupler strength, which can be divided into two key processes: mapping edge weight and determining coupling strength within subset.

3.2.1 Edge Weight Mapping

For each edge (i, j) in $E(I)$ with a weight of J_{ij}, the objective is to identify spins within $\phi(i)$ and $\phi(j)$ that are connected. Once such spins are found, the weight value J_{ij} is assigned to the edge connecting them. In cases where multiple edges connect $\phi(i)$ and $\phi(j)$, J_{ij} is assigned to one edge, while the remaining connecting edges receive a weight coefficient of 0. Note that there is always at least one valid edge, based on the definition of minor embedding.

3.2.2 Coupling Strength within Subset

The second component involves adding positive coupling strength between spins belonging to the same $\phi(i)$ subset. This additional strength ensures consistent spin values within each subset. To achieve this consistency, this coupling coefficient, denoted as F_{ii} must be set to a value that is not less than the upper bound of the sum of the absolute values of the coupling weight acting on the spin s_i in the input graph. This can be expressed as:

$$F_{ii} = F(i) = \begin{cases} \sum_{j=1}^{N} |J_{ij}| & \text{, if } \exists\, j \in \{1, \ldots, N\} \text{ such that } J_{ij} \neq 0, \\ 1 & \text{, if } J_{ij} = 0 \text{ for all } j \in \{1, \ldots, N\}. \end{cases} \tag{3.1}$$

Note that for any spin s_i within the input graph that has no external coupling, i.e., where $J_{ij} = 0$ for all j, a default positive value, such as 1, is assigned as its coupling coefficient within the subset $\phi(i)$. This assignment ensures that the spins in the subset exhibit consistent values. On the contrary, setting $F_{ii} = 0$ for such cases would result in inconsistency among the spin values within $\phi(i)$.

3.2.3 Shifted Ising Hamiltonian

The mapping process results in a shifted version of the Ising Hamiltonian of the input graph. Given the Ising Hamiltonian of the input graph:

$$H(\vec{s}) = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} J_{ij} s_i s_j. \tag{3.2}$$

Assuming that the King's graph is mapped using the strategy described in Section 3.1, and that the spins in each $\phi(i)$ have the same spin value s_i due to proper assignment on edges as introduced in Section 3.2, the Ising Hamiltonian of the King's graph can be derived as:

$$H(\vec{s}) = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} J_{ij} s_i s_j - \sum_{i=1}^{N} F_{ii}(num_i - 1), \tag{3.3}$$

where num_i is the number of vertices in the subset $\phi(i)$ and F_{ii} is the positive coupling strength between spins belonging to this subset $\phi(i)$. Consequently, the Ising Hamiltonian of the King's graph has a constant shift value compared to the one of the input graph.

3.3 Implementation and Simulation

Implementation of both the best-known complete graph embedding and the PSSA embedding, along with weight coefficient mapping, was carried out using Python. The Python script is designed to map an input graph provided in Rudy format onto a King's graph, with the width L specified as an input parameter. The resulting graph is also stored in Rudy format. The code is available in the King's Graph Mapping repository.

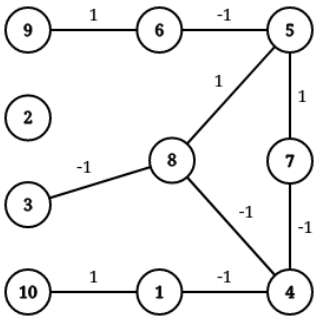
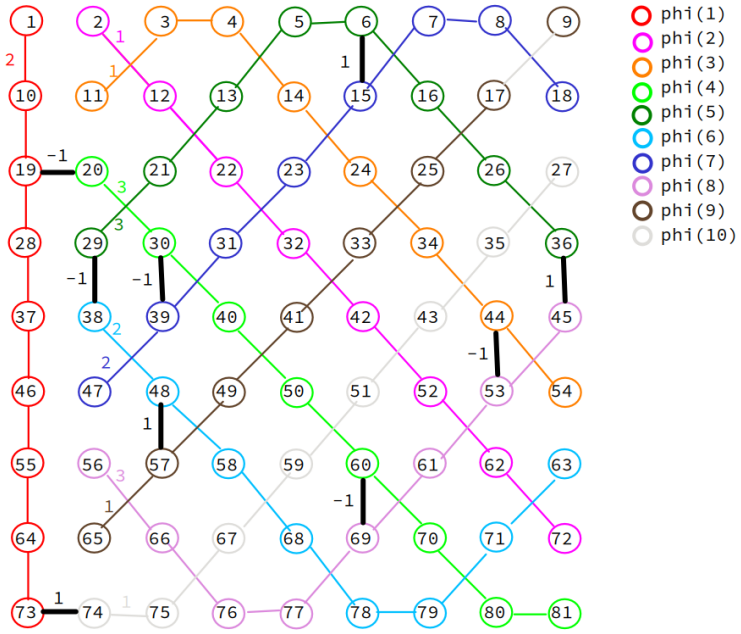
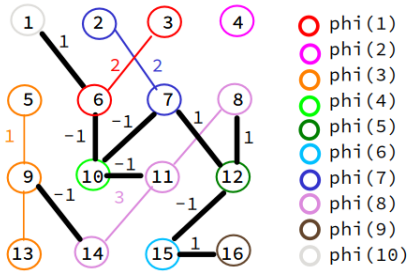


Figure 9: Graph J10_06.



(a) Graph J10_06 in $KG_{9,9}$ using the best-known complete graph mapping.



(b) Graph J10_06 in $KG_{4,4}$ using PSSA mapping.

Figure 10: King's graphs mapped from graph J10_06. Spins circled in the same color belong to the same subset ϕ, as indicated in the legend. Edges sharing the same color as the circles represent coupling edges within the same subset, with the corresponding coupling coefficient value F_{ii} labeled in the same color. Black-colored edges denote connections mapped from the input graph, with the value J_{ij} labeled.

3.3.1 A Mapping Example

To illustrate the mapping result, we took graph J10_06 as the input graph and performed both best-known complete graph mapping and PSSA graph mapping respectively. Graph J10_06, as shown in Fig. 9, consists 10 spins with a 20% edge density and uniform distribution weights in $\{-1, 1\}$. The width using best-known complete graph mapping was $N - 1 = 9$, while the minimum width achieved using PSSA mapping was 4. The resulting graph are shown in Fig. 10.

3.3.2 Simulations using SA and OIM

Simulations were conducted on three selected J10-set problems (J10_01, J10_06, and J10_11) and their corresponding mapped King's graphs with different widths, using simulated annealing (SA) algorithm and stochastic differential equation (SDE) model of OIM on C.

For SA, simulation parameters were set as follows: the number of iteration was 100000, and the temperature ranged from an initial value of 1 to a final value of 0.005 using a linear schedule. For OIM, simulation parameters were set as follows: the time step was 0.01, the stop time was 1000, Kc was fixed at 2.8, and Ks was defined as a pulse with a period of 10, bounded between 0.0 and 1.0. Each simulation method was executed 100 times with different random initial conditions for each graph.

The shift value of the minimum Ising Hamiltonian was determined during the mapping process using (3.3). The simulation results presented in table Table 3 indicates how many times the graph achieved its minimum Ising Hamiltonian. Additionally, a Python post-processing script was designed to verify whether the spins within the same subset ϕ had matching spin values, i.e., $s_k = s_l$ for all $s_k, s_l \in \phi(i), i = 1, ..., N$. It was confirmed that spin consistency was maintained among all simulations performed. Observing the simulation results in the table, it shows that the mapped graph achieved the minimum Ising Hamiltonian less frequently compared to the original graph as the number of spins in the mapping graph increased.

Table 3: Simulation results of original graph and mapped King's graph. The count is the number of times the simulation achieved its minimum Ising Hamiltonian among 100 runs.

(a) J10_01

J10_01	original	$KG_{4,4}$	$KG_{5,5}$	$KG_{6,6}$	$KG_{7,7}$
Hmin	-9	-27	-43	-68	-102
SA count(Hmin)	100	92	98	69	77
OIM count(Hmin)	100	100	100	64	55

(b) J10_06

J10_06	original	$KG_{4,4}$	$KG_{5,5}$	$KG_{6,6}$	$KG_{7,7}$
Hmin	-9	-22	-39	-56	-101
SA count(Hmin)	100	100	85	86	55
OIM count(Hmin)	100	100	100	54	40

(c) J10_11

J10_11	original	$KG_{4,4}$	$KG_{5,5}$	$KG_{6,6}$	$KG_{7,7}$
Hmin	-532	-1274	-2054	-4226	-4463
SA count(Hmin)	75	28	26	12	13
OIM count(Hmin)	87	100	12	0	0

4 OIM Chip Design

4.1 Introduction to OIM

Oscillator Ising Machines are networks formed with coupled self-sustaining nonlinear oscillators. With subharmonic injection locking introduced, the system naturally approaches the Ising Hamiltonian.

Throughout the past year, our team successfully taped out two distinct OIM chips, each featuring different architectures and applications. The first one is a 50-spin dense OIM (DOIM) chip, which has a fully connected coupling network, where each oscillator establishes coupling connections with 49 other oscillators on the chip. Next, the second chip is a 261-spin Field-Programmable Ising Machine (FPIM), which adopts a programmable sparse coupling network configuration, where the maximum number of coupling connections each oscillator can establish with other oscillators is 64. The 50-spin DOIM chip is designed to tackle MIMO-MLE problems [2], while the 261-spin FPIM chip is designed to address SAT20 problems [3].

4.2 50-spin DOIM Chip

The design of the 50-spin DOIM chip comprises two main blocks: the OIM block and the top-level controller. The OIM block contains oscillators, capacitor banks, miscellaneous analog blocks, and oscillator helper circuits, which collectively facilitate the functionality of the network. Conversely, the top-level controller is responsible for interfacing with desktop/laptop systems via a UART interface. Its primary functions include configuring the OIM coupling network to address various problems and retrieving the state of the OIM, including frequency, phase, and configuration register data.

The design stages of the DOIM chip encompass analog design, digital design, and mixed-signal integration. During the tapeout process, my primary responsibilities were centered on digital design tasks, with a specific focus on synthesis and place-and-route (PNR) activities. My contributions primarily involved modification and optimization of the PNR script. This entailed a comprehensive range of tasks aimed at enhancing the chip's layout and resolving specific design challenges.

- **Technology Files Setup**: Essential technology files in TSMC 28nm for synthesis and PNR scripts include
  - mmmc.tcl: Defines setup time and hold time analysis view for precise timing analysis. Required accompanying files:
    - Liberty (.lib) files: Contains logical descriptions and timing details for standard cells across different process, voltage, temperature (PVT) conditions.
    - QRC techfiles: Provides necessary information for creating resistance-capacitance (RC) corners, including interconnect models of metal layers, PVT variations, and calibration data.

[2] MIMO, specifically MU-MIMO (Multi-User Multiple-Input-Multiple-Output) decoding, is a crucial problem in wireless communication. In MU-MIMO scenarios, multiple users with multiple transmit antennas share the same resources to transmit signals to a receiver equipped with multiple receive antennas. Consequently, each received signal comprises a noisy combination of symbols transmitted by multiple users. The aim of MIMO-MLE (Maximum Likelihood Estimation) is to recover the most likely set of transmitted symbols from the received signals. [7]

[3] SAT (Boolean Satisfiability problem) is defined as follows: given a boolean formula F(x1, ..., xn), if F can be evaluated to 1 (true), it is considered satisfiable. The SAT problem involves finding the values of variables (xi's) that satisfy F. In the case of SAT20, "20" indicates the number of variables in the formula F.

– synthesis/pnr.tcl: Establishes synthesis/PNR options and executed synthesis/PNR commands. Required accompanying files:
  * Technology Library Exchange Format (.tlef) file: Contains physical properties of all layers/vias and design rules related to their geometry.
  * Library Exchange Format (.lef) file: Contains physical properties of standard cells within the library.

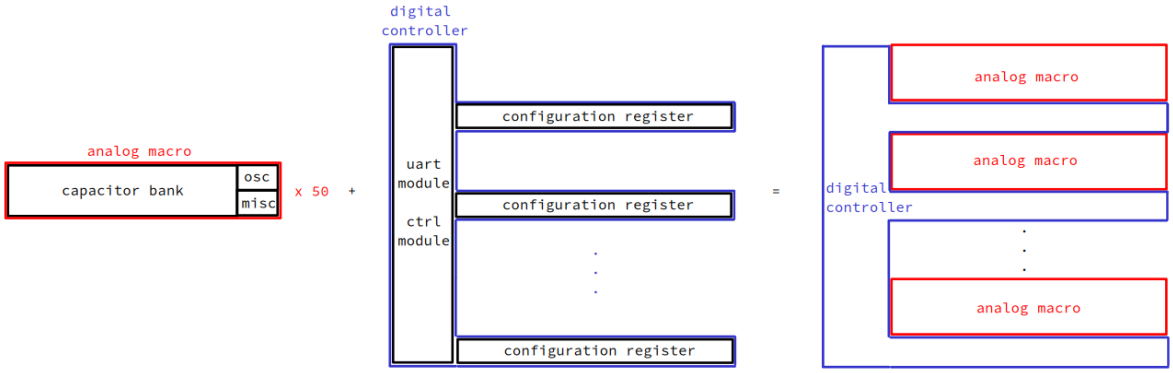


Figure 11: Illustration of the floorplan. Red block is the analog macro, including oscillator, capacitor bank, and miscellaneous blocks. Blue block is the digital controller, including UART module, controller module, and configuration registers.

- **Floorplan**: The analog macro, comprising oscillators, capacitor banks, and miscellaneous analog blocks, was arranged in a 1-D vertical array, while the digital top-level controller positioned to the left of the array, as shown in Fig. 11.
  – *Routing Blockage*: To prevent unwanted routing, routing blockages were strategically placed to the right of each analog macro, allowing only power routing in that area. Example command: `createRouteBlk -layer 1 3 4 5 6 7 8 9 -box 1903.35 31.67 1909.88 61.1`.
  – *Power Pin Placement*: Power pins were added using the command `createPGPin`. This step is essential for future Layout versus Schematic (LVS) comparisons. Since the standard cells in the schematic exported from the PNR results include power pins, integrating a global power pin is important to connect them all to the top level. Example command: `createPGPin VDD -net VDD -geom M2 3.2 10.0 3.6 10.2`.
  – *Controller Placement*: Originally, without any specified constraint, the controller was positioned in the narrow space between the analog macros, causing crowded cell placement and routing difficulties due to limited space. Collaborating with Shreesha, we addressed this issue by using the command *createGuide*. This command allowed us to specify the preferred placement of the controller module to the left side of the entire chip, away from the analog macros. This strategy eliminated crowded placement and routing challenges. Example command: `createGuide myCtrlGroup 5 5 392.63 293.0`.
- **Power Planning**: The power planning process was divided into two stages. The first stage fell within the scope of PNR on Innovus, including the setup and placement of core ring (M1 and M2), block ring (M1 and M2), power stripe (M3 to M7), and standard cell

power rail (M1). Notably, the placement of power stripes was optimized to mitigate DRC violations. For example, M3 stripe often caused difficulty in routing M2 or M3 wires near standard cell pins, resulting in DRC violation. To address this issue, M3 stripes were specifically placed on top of well taps, where no routing was present, solving the recurring DRC violation problem. Following the power planning phase on Innovus, the second stage involved global power routing using the topmost metal layers (M8 and M9) on Virtuoso, which was undertaken by other team members.

- **Timing Constraint**: Insufficiently defined timing constraints in the early stages resulted in difficulties with PNR, especially encountering stalls in the Innovus scripts during placement or routing steps. The limited space beneath the analog macros in the 1-D array magnified these challenges. Therefore, Shreesha and I dedicated significant effort to meticulously establish clock and pin constraints to alleviate these issues.
  - *Input Capacitance and Output Load*: Input capacitance and output load of the chip are set by the commands: `set_max_capacitance cap_value input_pin` and `set_load cap_value output_pin`.
  - *Delay Through a Module*: To specify a desired delay through a module, i.e., the delay passing from input signals to output signals of a module, we need to use the command: `set_max_delay delay_value -through my_module`. We used this command to specified the delay through the oscillator helper, which configures the coupling on oscillator network.
  - *Timing Information of a Blackbox*: There are two ways we tried to specify the input capacitance and output load of a blackbox, which is the analog macro in our case. The first method was using the command `set_load -max/min cap_value net` to overwrite the maximum or minimum capacitance on a net. More formally, the second method was writing a LIB file for analog macro. In the LIB file, we can provide information on every I/O pin regarding its I/O direction, normal/rise/fall capacitance of input pin, max/min capacitance of output pin. The format would be as follows:

    ```
    pin(input_pin_name) {
      direction:input; capacitance:cap_value;
      rise_capacitance:cap_value; fall_capacitance:cap_value;
    }
    pin(output_pin_name) {
      direction:output;
      max_capacitance:cap_value; min_capacitance:cap_value;
    }.
    ```
  - *Verification of Constraints on Innovus*: After PNR completed, some commands were utilized to verify the capacitance and delays of the final layout were matched with the constraints we set. Example command for reporting delay: `report_timing -from net -to net -point_to_point`. Example command for reporting capacitance on analog macro's pin: `report_property [get_nets [get_property [get_pins module/pin_name] net_name]]`.

4.3 261-spin FPIM Chip

The architecture of the 261-spin FPIM chip is comprised of three primary elements: the OIM block, the FPIM architecture, and the top-level controller.

Same as the previous chip, the OIM block contains oscillators, capacitor banks, miscellaneous analog blocks, and oscillator helper circuits.

The FPIM architecture, proposed in [3], introduces tiles as units to form the field-programmable fabric matrix. Each tile contains a switch block, x-direction and y-direction connection blocks, and an Ising logic block, which corresponds to the OIM block in the previous content. In this chip, the tiles were arranged in a 9-row by 29-column array, while the top-level controller positioned to the left of this 2-D array.

The top-level controller's functionality includes setting/getting the value of configuration registers and reading frequency or phase of oscillators. These configuration registers, located within each tile, set the state of tri-state buffers in switch block and connection blocks, as well as the parameters of the oscillator helper. Due to the substantial number of configuration registers (up to 40,000 per tile row), directly connecting each register in the tiles to the digital controller would result in an excessive number of routings, which is not desirable. To address this issue, the controller utilized a streaming method for programming/reading configuration registers, allowing only a single routing path needed from the controller to each tile row for configuration register-related operations.

My primary responsibility during the design process of this chip centered around the design and verification of the digital top-level controller. The tasks I completed include:

- **System Integration**: Discussed with Shreesha, Pavan, and Thomas regarding signal transmission between the tile array and the controller, handling operations such as frequency reading, phase reading, and programming/reading of configuration registers. Meanwhile, I maintained diagrams that illustrated implementation strategies for these features within the tile array, as shown in Fig. 12.

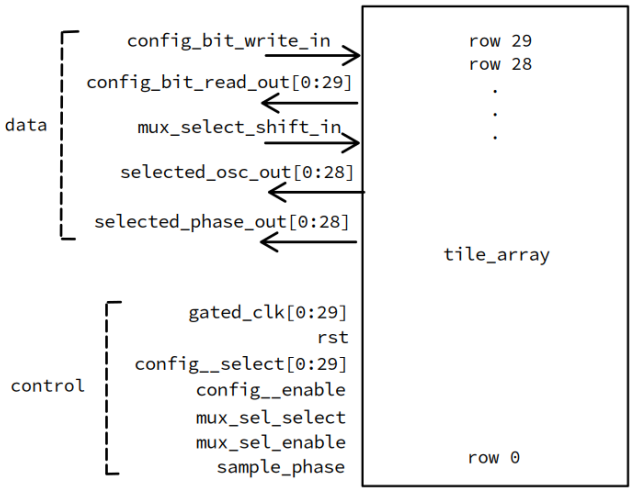
- **Clock Gating**: The clock gating implementation flow, from RTL to synthesis and PNR, was established. An example of RTL coding style recognizable as clock gating by Genus, the synthesis tool, is as follows:

```
always @(posedge clk) begin
        if(rst) Q <= 1'b0;
        else begin
                if(enable) Q <= D;
                else Q <= Q;
        end
end
```

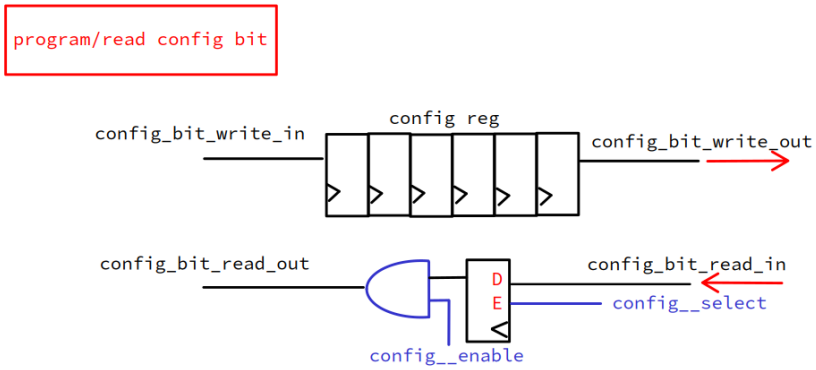
  To enable clock gating, the following commands need to be added in the synthesis script:
  `set_db lp_insert_clock_gating true`
  `set lp_clock_gating_cell [ list Gated_Clock_Latch_Standard_Cell_Names ]`
  For PNR, no additional command is needed to establish clock gating, and clock gating implementation can be observed in the post-PNR results using the clock tree debugger.

- **Controller RTL Design**: The top-level controller was designed in Verilog. Equipped with UART interfaces, the controller receives commands from external sources and executes corresponding functions, such as streaming in/out the configuration bit values into specific tile rows. The RTL code is parameterized, allowing it to adapt to tile arrays with varying numbers of rows or columns, with all parameters read from a file named "defines.v" to ensure clarity and maintainability. A comprehensive documentation outlining the design and functionality of the top-level controller is included in Appendix C.

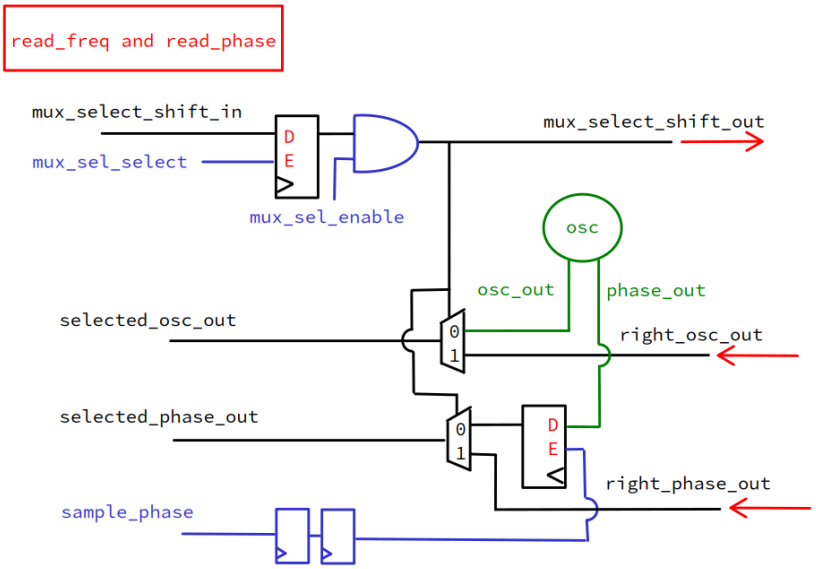
- **Testbench Design**: Testbenches for the top-level controller module and its submodules were designed. The testbench for the top-level controller module integrates comprehensive test cases utilizing the UART protocol to simulate real-world scenarios. These test



(a) Illustration depicting the I/O data and control signals on the tile array interface.



(b) Diagram showcasing the logic used in each tile for programming and reading operations on configuration registers.



(c) Diagram demonstrating the logic implemented in each tile for frequency and phase reading.

Figure 12: Interface and logic implementations in tile array.

cases encompass a wide range of scenarios, including exhaustive coverage of all possible commands or extensive testing with random inputs. During testing, the output data or the state of the design under test (DUT) were meticulously compared to the expected golden data to verify accuracy and functionality. For instance, when programming values into the tile array, the testbench validates the programmed configuration registers by comparing their bit-by-bit values with the golden data after receiving confirmation of successful programming. Examples of the testbench is provided in Appendix D.

- **Simulations**: Simulations were conducted at various stages, including RTL, post-synthesis, and post-PNR simulations, utilizing VCS. Accurate timing simulation after synthesis or PNR required properly annotated Standard Delay Format (SDF) files. This annotation process involved two steps: (1) adding a command inside the testbench to annotate the SDF file with hierarchical path names of the modules, for example:

```
initial begin
$sdf_annotate(sdf_file, hierarchical_path_name_of_the_module);
end
```

  and (2) specifying options on the VCS command line, such as `+compsdf +mindelays` or `+compsdf +maxdelays`.

5 Conclusion

In this study, we have delved into the Bistable Latch Ising Machine (BLIM), the mapping of Ising problems onto King's graph, and the design process of Oscillator Ising Machine (OIM) chips.

In Section 2, we represented the fundamental theory of BLIM (Section 2.1), and highlighted its efficacy in solving small-scale Ising problems through both simulations (Section 2.2) and measurements (Section 2.3). While our findings underscore BLIM's promise, opportunities for refinement and validation remain, particularly in distinguishing the difference between single-sided and double-sided coupling in theory, exploring alternative simulation inputs, and conducting measurements on larger problem sets.

In Section 3 detailed the process of mapping Ising problems onto King's graphs. This process, including finding a minor embedding (Section 3.1) and edge weight mapping (Section 3.2), has been elaborated and implemented (Section 3.3). Although the simulation results indicate a decrease in the frequency of achieving the minimum Ising Hamiltonian for these mapped problems, there remains a possibility for improvement. By adopting different simulation setups on original graph and mapped graph, we might uncover new insights into the performance and applicability of King's graph mapping.

In Section 4, we focused on the design process of OIM chips. Spanning across the development of both DOIM and FPIM chips, our efforts covered RTL design, synthesis, and PNR, along with overcoming various challenges encountered during the development phase.

In conclusion, our exploration of BLIM, King's graph mapping, and OIM chip design highlights our commitment to pushing the boundaries of unconventional computational strategies.

6 References

[1] Roychowdhury, J. (2021). Bistable Latch Ising Machines. In: Kostitsyna, I., Orponen, P. (eds) Unconventional Computation and Natural Computation. UCNC 2021. Lecture Notes in Computer Science(), vol 12984. Springer, Cham. https://doi.org/10.1007/978-3-030-87993-8_9

[2] Wang, T., Roychowdhury, J. (2019). OIM: Oscillator-Based Ising Machines for Solving Combinatorial Optimisation Problems. In: McQuillan, I., Seki, S. (eds) Unconventional Computation and Natural Computation. UCNC 2019. Lecture Notes in Computer Science(), vol 11493. Springer, Cham. https://doi.org/10.1007/978-3-030-19311-9_19

[3] Jagielski, T., Manohar, R., & Roychowdhury, J. (2023). FPIM: Field-Programmable Ising Machines for Solving SAT. CoRR, abs/2306.01569. https://doi.org/10.48550/ARXIV.2306.01569

[4] Sugie, Y., Yoshida, Y., Mertig, N. et al. Minor-embedding heuristics for large-scale annealing processors with sparse hardware graphs of up to 102,400 nodes. Soft Comput 25, 1731–1749 (2021). https://doi.org/10.1007/s00500-020-05502-6

[5] Okuyama T, Yoshimura C, Hayashi M, Tanaka S, Yamaoka M (2016) Contractive graph-minor embedding for CMOS Ising computer. IEICE Tech Rep 116:97–103

[6] Choi V (2008) Minor-embedding in adiabatic quantum computation: I. The parameter setting problem. Quantum Inf Process 7(5):193– 209

[7] Roychowdhury, J., Wabnig, J., Srinath, K. P., et al. (2021, September 22). Performance of Oscillator Ising Machines on Realistic MU-MIMO Decoding Problems (Version 1) [Preprint]. Research Square. https://doi.org/10.21203/rs.3.rs-840171/v1

A Proof of Coupling Latch Network Systems

A.1 Recapitulation of Proof for Positive Coupling Latch Network

To prove that the coupled latch networks in Section 2 function as Ising machines, we need to derive the Lyapunov function of the network and demonstrate that it is identical to the Ising Hamiltonian under appropriate conditions [1].

In [1], the derivation of positive coupling latch networks has been established. For a BLIM with N coupling latches, a system of N differential equations results

$$\frac{dv_i(t)}{dt} = f(v_i;k) - \sum_{j=1}^{N} J_{ij}\ g(v_i,v_j;k),\ \ i=1,\ldots,N, \tag{A.1}$$

where

$$f(v_i;k) = \frac{G}{C}(\tanh(k\tanh(kv_i)) - v_i), \tag{A.2a}$$

$$g(v_i,v_j;k) = \frac{v_i - v_j}{C}, \tag{A.2b}$$

$$G = \frac{1}{R_o}, \tag{A.2c}$$

$$\text{and } Jij = \frac{1}{R_{ij}}. \tag{A.2d}$$

Its Lyapunov function is defined as

$$L_N(\vec{v};k) = -\left(\sum_{i=1}^{N} z(v_i;k) - \sum_{i,j=1}^{N} J_{ij}h(v_i,v_j;k)\right). \tag{A.3}$$

where

$$z(v;k) = \int_0^v f(x;k)\,dx\ , \tag{A.4a}$$

$$\text{and } h(v_i, v_j; k) = \frac{1}{2C}[\frac{(v_i - v_j)^2}{2} - 1]. \tag{A.4b}$$

With the following assumptions confirmed, it is proven that the Lyapunov function equals a scaled/shifted Ising Hamiltonian at bistable values. This implies that there exists a total order correspondence between the Hamiltonian and Lyapunov functions, and that the Hamiltonian and Lyapunov global minima correspond under bistability.

1. $f(\cdot)$ is the derivative of $z(\cdot)$: $f(v_m;k) = \frac{dz(v_m;k)}{dv_m}, m = 1, \ldots, N$.
2. $h(\cdot)$ and $g(\cdot)$ are related as: $g(v_m, v_j, \text{sgn}(J_{ij}); k) = \frac{\partial h(v_m, v_j, \text{sgn}(J_{ij}); k)}{\partial v_m} + \frac{\partial h(v_j, v_m, \text{sgn}(J_{ij}); k)}{\partial v_m}$.
3. $f(v;k)$ is bistable if $k = K$, for some sufficiently large gain $K > 0$, i.e., for some v_+, v_-, with $v_+ > v_-$, $f(v_+;K) = f(v_-;K) = 0$. Moreover, $\frac{df(v;K)}{dv}|_{v=v_+} < 0$ and $\frac{df(v;K)}{dv}|_{v=v_-} < 0$.
4. $z(v_+;K) = z(v_-;K) = c_3$, for some value c_3.
5. $h(v_+, v_+, \text{sgn}(J_{ij}); K) = h(v_-, v_-, \text{sgn}(J_{ij}); K) = c_1$, $h(v_+, v_-, \text{sgn}(J_{ij}); K) = h(v_-, v_+, \text{sgn}(J_{ij}); K) = c_2$, for some values c_1 and $c_2 > c_1$.

A.2 Derivation for Negative Coupling Latch Network

Now, in consideration of a more general coupling network that includes negative couplings, we proceed to re-derive the system's differential equation and the Lyapunov function. Subsequently, we aim to validate the assumptions made earlier to demonstrate that the system, comprising both positive and negative coupling latches, functions as an Ising machine.

A.2.1 KCL Differential Equations

Using the latch model and the positive/negative coupling configurations as defined in Fig. 1 and Fig. 2 respectively, we consider a network comprised of N coupling latches. Given a coupling between the i^{th} and j^{th} latches, the KCL equation at node v_{i+} can be expressed as

$$C\frac{dv_{i+}(t)}{dt} = \begin{cases} \frac{\tanh(-kv_{i-})-v_{i+}}{R_o} - \frac{v_{i+}-v_{j+}}{R_{ij}}, \text{ if } J_{ij} > 0, \\ \frac{\tanh(-kv_{i-})-v_{i+}}{R_o} - \frac{v_{i+}-v_{j-}}{R_{ij}}, \text{ if } J_{ij} < 0. \end{cases} \tag{A.5}$$

Applying the inverter model (2.2) to replace v_{i-} and v_{j-}, and denoting v_{i+} as v_i, the equation becomes

$$\frac{dv_i(t)}{dt} = \begin{cases} \frac{1}{C}\left[\frac{\tanh(k\tanh(kv_i))-v_i}{R_o} - \frac{v_i-v_j}{R_{ij}}\right], \text{ if } J_{ij} > 0, \\ \frac{1}{C}\left[\frac{\tanh(k\tanh(kv_i))-v_i}{R_o} - \frac{v_i-\tanh(-kv_j)}{R_{ij}}\right], \text{ if } J_{ij} < 0. \end{cases} \tag{A.6}$$

To support both types of couplings, equations (A.2a) and (A.2c)can be reused, while equations (A.2b) and (A.2d) are adjusted as follows.

$$g(v_i, v_j, \text{sgn}(J_{ij}); k) = \begin{cases} \frac{1}{K_c C}(v_i - v_j) & \text{, if } J_{ij} > 0, \\ \frac{-1}{K_c C}(v_i - \tanh(-kv_j)) & \text{, if } J_{ij} < 0, \end{cases} \tag{A.7a}$$

$$|Jij| = \text{sgn}(J_{ij})J_{ij} = \frac{K_c}{R_{ij}}. \tag{A.7b}$$

A more general form for N latches with a coupling network that is built according to a set of J_{ij} can thus be derived as

$$\frac{dv_i(t)}{dt} = f(v_i;k) - \sum_{j=1}^{N} J_{ij}\ g(v_i, v_j, \mathrm{sgn}(J_{ij});k),\ \ i = 1,\ldots,N. \tag{A.8}$$

A.2.2 Lyapunov Function

As demonstrated in [1],

$$L_N(\vec{v};k) = -\left(\sum_{i=1}^{N} z(v_i;k) - \sum_{i,j=1}^{N} J_{ij}h(v_i,v_j;k)\right) \tag{A.9}$$

is a Lyapunov function if the following equations hold and in case of symmetric coupling.

$$f(v_m;k) = \frac{dz(v_m;k)}{dv_m}, m = 1,\ldots,N, \tag{A.10a}$$

$$g(v_m, v_j, \mathrm{sgn}(J_{ij});k) = \frac{\partial h(v_m, v_j, \mathrm{sgn}(J_{ij});k)}{\partial v_m} + \frac{\partial h(v_j, v_m, \mathrm{sgn}(J_{ij});k)}{\partial v_m}. \tag{A.10b}$$

Therefore, assuming that $z(\cdot;\cdot)$ and $h(\cdot;\cdot)$ satisfy the conditions, we have (A.9) as a Lyapunov function for (A.8).

A.2.3 Lyapunov-Ising Hamiltonian Relation

In this section, we explore the relationship between the Lyapunov function and the Ising Hamiltonian, ultimately leading to the conclusion that the Lyapunov function equals a scaled/shifted Ising Hamiltonian.

Referring to [1], the (discrete) Ising Hamiltonian of a system with N spins $\{s_i\}$ and a set of symmetric coupling weights J_{ij}, is defined as

$$H(\vec{s}) = -\frac{1}{2}\sum_{i,j=1}^{N} J_{ij}s_is_j, \tag{A.11}$$

where $s_i = +1$ or -1. Moreover, it is noted that a scaled/shifted version of the Ising Hamiltonian maintains total order.

It has been demonstrated that the Lyapunov function equals a scaled/shifted Ising Hamiltonian under following assumptions [1]:

- **Assumption 1**. For each latch, v_+ and v_-, with $v_+ > v_-$, are the only stable equilibria:

  We assume $f(v;k)$ is bistable if $k = K$, for some sufficiently large gain $K > 0$, i.e., for some v_+, v_-, $f(v_+;K) = f(v_-;K) = 0$, $\frac{df(v;K)}{dv}|_{v=v_+} < 0$ and $\frac{df(v;K)}{dv}|_{v=v_-} < 0$.

- **Assumption 2**. At the bistable values v_+ and v_-, $z(\cdot;\cdot)$ and $h(\cdot;\cdot)$ have properties:

$$z(v_+;K) = z(v_-;K) = c_3, \tag{A.12a}$$

$$h(v_+, v_+, \mathrm{sgn}(J_{ij});K) = h(v_-, v_-, \mathrm{sgn}(J_{ij});K) = c_1, \tag{A.12b}$$

$$h(v_+, v_-, \mathrm{sgn}(J_{ij});K) = h(v_-, v_+, \mathrm{sgn}(J_{ij});K) = c_2, \tag{A.12c}$$

  for some values c_1, $c_2 > c_1$, and c_3.

Therefore, the Lyapunov function in (A.9) can be viewed as a scaled/shifted Ising Hamiltonian, if both assumptions 1 and 2 can be validated under negative coupling network. We will now proceed to verify these assumptions.

Following [1], we choose $K = 5$ as the high value of gain , and define a spin as:

$$s_i = \begin{cases} 1 & \text{if } v_i = v_+, \\ -1 & \text{if } v_i = v_-. \end{cases} \tag{A.13}$$

- **Proof for Assumption 1**: The function $f(\cdot)$ remains unchanged regardless the presence of negative coupling. Thus, as demonstrated in [1], solving $f(v; K) = 0$ yields

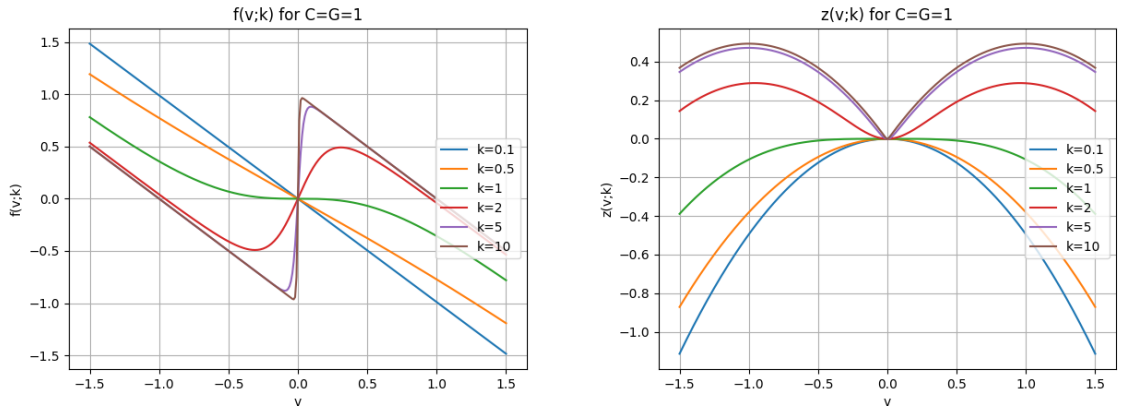
$$v_+ \simeq +1, \; v_- \simeq -1, \; \frac{df}{dv}(v_+) < 0, \; \frac{df}{dv}(v_-) < 0,$$

  which satisfies assumption 1.

- **Proof for (A.12a) in Assumption 2**: Given that our function $f(\cdot)$ remains consistent with that in [1], we can adopt the same definition for $z(\cdot;\cdot)$:

$$z(v; k) = \int_0^v f(x; k)\, dx \; . \tag{A.14}$$

  , which clearly satisfies the assumption stated in Appendix A.2.2 that equation (A.10a) must hold. As shown in Fig. 13, since $f(v; k)$ is odd in v, $z(v; k)$ is even, satisfying (A.12a).



(a) $f(v; k)$. (b) $z(v; k)$.

Figure 13: The waveform of the functions $f(\cdot)$ and $z(\cdot)$ in BLIM system.

- **Proof for (A.12b) and (A.12c) in Assumption 2**: To begin with, we want to simplify function $g(\cdot;\cdot)$ by replacing its nonlinear element $tanh(\cdot)$. With a large $k = K = 5$, the spin voltages v_i become bistable, i.e., $v_i \in \{v_+, v_-\}$, $v_+ \simeq +1$, and $v_- \simeq -1$. We approximate $tanh(\cdot)$ at these bistable voltages using

$$\tanh(-Kv_+) = v_- = -v_+, \; \tanh(-Kv_-) = v_+ = -v_-, \tag{A.15}$$

  as shown in Fig. 14.

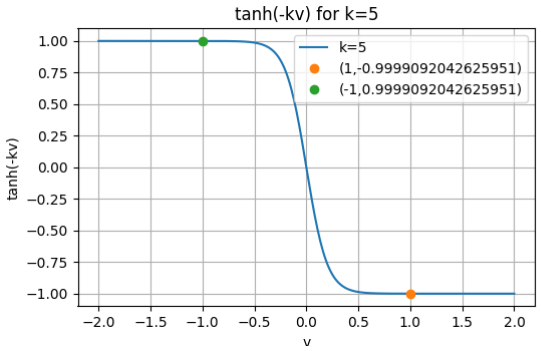


Figure 14: The function $\tanh(\cdot)$ with a large enough $k = K = 5$, $v_+ = +1$ and $v_- = -1$.

Therefore, (A.7a) can be rewritten as

$$g(v_i, v_j, \mathrm{sgn}(J_{ij}); K) = \frac{-1}{K_c C}(v_i - \tanh(-K v_j)) = \frac{-1}{K_c C}(v_i + v_j), \text{ if } J_{ij} < 0. \tag{A.16}$$

The graphs of the function $g(v_i, v_j, \mathrm{sgn}(J_{ij}) = -1; k)$ are shown in Fig. 15a, using both the original $\tanh(\cdot)$ model and the approximated models with varying values of k . Fig. 15b shows the overlap between the $\tanh(\cdot)$ model and the approximated model with a large gain $k = 5$ from a side view. Note that the waveform overlaps when the spin voltages are bistable. In contrast, Fig. 15c shows the overlap between the models with a small gain $k = 1$, showcasing a rough match when the spin voltages are small but lacking overlap when the voltages are bistable.

Now We define

$$h(v_i, v_j, \mathrm{sgn}(J_{ij}); K) = \frac{-1}{2K_c C}\left[\frac{(v_i + v_j)^2}{2} - 1\right], \text{ if } J_{ij} < 0, \tag{A.17}$$

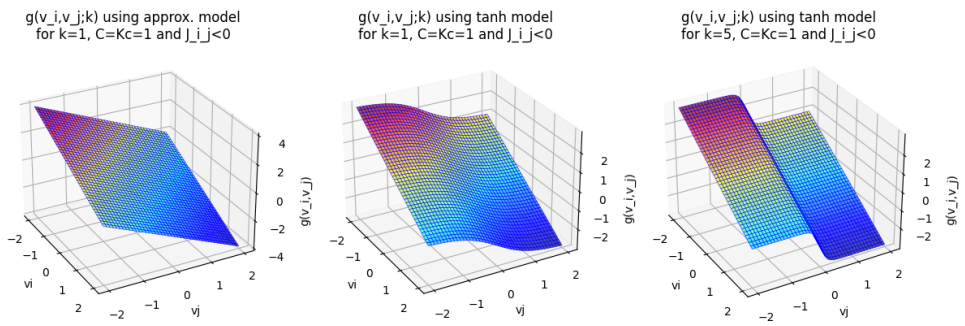
which satisfies the assumption stated in Appendix A.2.2 that equation (A.10b) hold with

$$\begin{aligned}\frac{\partial h(v_m, v_j, \mathrm{sgn}(J_{ij}); k)}{\partial v_m} + \frac{\partial h(v_j, v_m, \mathrm{sgn}(J_{ij}); k)}{\partial v_m} &= \frac{-1}{2K_c C}[v_i + v_j] + \frac{-1}{2K_c C}[v_i + v_j] \\ &= \frac{-1}{K_c C}[v_i + v_j] \\ &= g(v_m, v_j, \mathrm{sgn}(J_{ij}); k), \text{ if } J_{ij} < 0.\end{aligned} \tag{A.18}$$

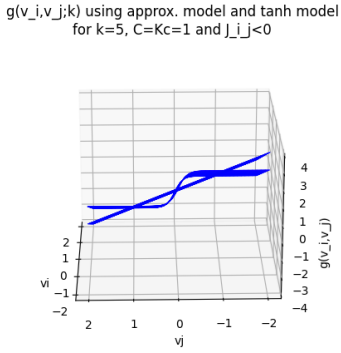
Moreover, (A.12b) and (A.12c) are also satisfied with

$$\begin{aligned}c_1 &= h(v_+, v_+, J_{ij} < 0; K) = h(v_-, v_-, J_{ij} < 0; K) \\ &= -\frac{1}{2K_c C}(2v_+^2 - 1) \simeq -\frac{1}{2K_c C}\ , \\ c_2 &= h(v_+, v_-, J_{ij} < 0; K) = h(v_-, v_+, J_{ij} < 0; K) \\ &= \frac{1}{2K_c C} > c_1.\end{aligned} \tag{A.19}$$

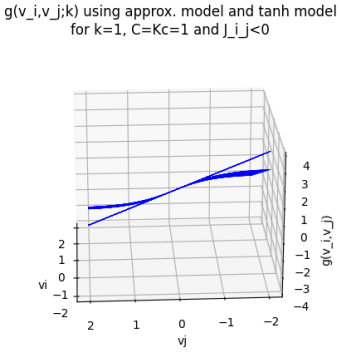
With all the assumptions satisfied, the Lyapunov function of a BLIM with both positive and negative couplings can be viewed as a scaled/shifted Ising Hamiltonian.



(a) $g(v1, v2; k)$ using the approximated model, tanh$(\cdot)$ model with k=1, and tanh$(\cdot)$ model with k=5, arranged from left to right.



(b) A side-view comparison between the approximated model and the tanh$(\cdot)$ model with $k = 5$.



(c) A side-view comparison between the approximated model and the tanh$(\cdot)$ model with $k = 1$.

Figure 15: Graph of $g(\cdot;\cdot)$ when $J_{ij} < 0$.

B Ising Problems used in this report

The 2/3/4/5-spin problems used in Section 2 are defined in Table 4, while J-10/20 problems are generated using rudy generator (http://web.stanford.edu/ yyye/yyye/Gset/). The density of all J-set problem is set as 20% and the randseed is 10000+ `suffix`, while the suffix of the problem name is corresponding to the following parameter setup:

- `J*_01` to `J*_05`: weight $(J_{ij}) = 1$.
- `J*_06` to `J*_10`: weight $(J_{ij}) = \{-1, 1\}$.
- `J*_11` to `J*_15`: weight $(J_{ij}) = \{-100, -99, ..., 0, ..., 99, 100\}$.

Table 4: List of Ising problem used in this document.

problem name	{i, j, Jij}	bruteforce min(Hamiltonian)	spin value for min(H) $\vec{s_i} = \{s_1, s_2, ..., s_n\}$
`size2_1`	{1, 2, +1}	-1	{+1, +1}
`size2_2`	{1, 2, -1}	-1	{-1, +1}
`size3_1`	{1, 2, +1}, {1, 3, +1}, {2, 3, +1}	-3	{+1, +1, +1}
`size3_2`	{1, 2, -1}, {1, 3, +1}, {2, 3, -1}	-3	{+1, -1, +1}
`size3_3`	{1, 2, +1}, {1, 3, -1}, {2, 3, +1}	-1	{+1, +1, +1} or {-1, +1, +1} or {-1, -1, +1}
`size4_1`	{1, 2, +1}, {1, 3, +1}, {2, 3, +1}, {2, 4, -1}, {3, 4, -1}	-5	{-1, -1, -1, +1}
`size4_2`	{1, 2, +1}, {1, 3, -1}, {1, 4, +1}, {2, 4, +1}, {3, 4, -1}	-5	{+1, +1, -1, +1}
`size5_1`	{1, 2, +1}, {1, 3, -1}, {1, 5, -1}, {2, 3, +1}, {2, 4, -1}, {2, 5, +1}, {3, 4, -1}, {4, 5, -1}	-6	{-1, +1, +1, -1, +1}

C Documentation of FPIM Controller

C.1 Commands List

1. **Init UART_CLK_CYCLES_PER_BIT**: Initializes UART clock cycles per bit rate. This command is sent only once at the beginning.
2. **Read Frequency**: Reads the frequency of an oscillator.
3. **Read Phase**: Reads the phase of all oscillators.
4. **Program Config Bit**: Programs configuration bit into a row.
5. **Read Config Bit**: Reads configuration bit from a row.
6. **Enable Config Bit**: Enables configuration bit of all tiles.
7. **Disable Config Bit**: Disables configuration bit of all tiles.

8. **Reset All**: Resets configuration bits of all tiles.

C.2 How to Use the Controller

To interact with the controller via UART, follow these steps:

1. **Connect to the Controller**: Establish a UART connection with the controller.
2. **Initialize the Controller**: Before sending any commands, the user needs to update the UART clock cycles per bit rate. The controller initially uses a value of 5000000/19200 = 260. Even if the desired value matches the default value, the user still needs to send the value to the controller.
3. **Select Command**: Choose the desired command from the list provided above.
4. **Format the Command**: Construct the command according to the specified format detailed below.
5. **Send Command**: Transmit the formatted command to the controller.
6. **Receive Response**: Await the response from the controller.
7. **Process Response**: Interpret the response received from the controller based on the expected format and content.
8. **Repeat Steps 3 to 7**: Continue interacting with the controller by repeating steps 3 to 7 as needed for additional commands or operations.

C.3 Command Format and Expected Response

This section provides a parametric representation of the command format and expected response using parameter names in defines.v attached in the end.

1. **Init UART_CLK_CYCLES_PER_BIT**:
   - Send Command:
     - Command Length: BYTE_UART_CLK_CYCLES_PER_BIT * 8 bits = W_UART_CLK_CYCLES_PER_BIT fill up to byte.
     - Command Format:
       - command[W_UART_CLK_CYCLES_PER_BIT-1:0] = UART clock cycles per bit
   - Receive Response:
     - Response Length: same as its command length
     - Response Format:
       - response[W_UART_CLK_CYCLES_PER_BIT-1:0] = UART clock cycles per bit. This value should be the same as what was previously sent.
2. **Read Frequency**:
   - Send Command:
     - Command Length: 24 bits
     - Command Format:
       - command[FLD_CMD_TYPE] = CMD_READ_FREQ

        * command[FLD_FRQ_CAL] = Frequency calculation granularity
        * command[FLD_OSC_COL_ID] = Oscillator/tile column index
        * command[FLD_OSC_ROW_ID] = Oscillator/tile row index
    - Receive Response:
      - Response Length: BYTE_FREQ_COUNTER * 8 bits = W_FREQ_COUNTER fill up to byte
      - Response Format:
        * response[W_FREQ_COUNTER-1:0] = Frequency calibration counts. This count represents the number of cycles of the selected oscillator signal during when the input clock of the chip had 2$\hat{}$Frequency_calculation_granularity cycles.

3. **Read Phase**:

    - Send Command:
      - Command Length: 8 bits
      - Command Format:
        * command[FLD_CMD_TYPE] = CMD_READ_PHASE
    - Receive Response:
      - Response Length: BYTE_OSC_ONE_HOT * 8 bits = NUM_OSC fill up to byte
      - Response Format:
        * response[NUM_OSC-1:0] = Phase detection output of each oscillator. The ith bit stands for the phase of the ith oscillator.

4. **Program Config Bit**:

    - Send Command:
      - Command Length: 8 bits
      - Command Format:
        * command[FLD_CMD_TYPE] = CMD_PRGM_CONFIG
        * command[FLD_CONFIG_ROW_IDX] = Oscillator/tile row index
    - Send Config Bit Values:
      - For normal rows:
        * Data Length: NUM_CONFIG_BITS_PER_ROW fill up to byte
        * Data Format:
          · data[NUM_CONFIG_BITS_PER_ROW-1:0] = {config_bit_tile_col_0, ..., config_bit_tile_col_last, config_bit_east_tile}
          · config_bit_tile_col_i = {sbox_0, ... sbox_last, cbox_x_in_0, ..., cbox_x_in_last,cbox_y_in_0, ..., cbox_y_in_last, cbox_x_out_0, ..., cbox_x_out_last, analog_0, ..., analog_last} corresponding to the tile in the ith column starting from the left
          · config_bit_east_tile = {sbox_0, ... sbox_last, cbox_y_0, ... c_box_y_last}
      - For the north normal:
        * Data Length: NUM_CONFIG_BITS_PER_NORTH_ROW fill up to byte
        * Data Format:

· data[NUM_CONFIG_BITS_PER_NORTH_ROW-1:0] = {config_bit_north_tile_col_0, ..., config_bit_north_tile_col_last, config_bit_northeast_tile}
· config_bit_northtile_col_i = {sbox_0, ... sbox_last, cbox_x_in_0, ..., cbox_x_in_last} corresponding to the tile in the ith column starting from the left
· config_bit_northeast_tile = {sbox_0, ... sbox_last}

- Receive Response:
  - Response Length: 8 bits
  - Response Format:
    * response[7:0] = 0. The response is used to indicate the operation is completed.

5. **Read Config Bit**:

- Send Command:
  - Command Length: 8 bits
  - Command Format:
    * command[FLD_CMD_TYPE] = CMD_READ_CONFIG
    * command[FLD_CONFIG_ROW_IDX] = Oscillator/tile row index
- Receive Response:
  - The response here is the config bits read out of the row.
  - For normal rows:
    * Response Length: NUM_CONFIG_BITS_PER_ROW fill up to byte
    * Response Format:
      · response[NUM_CONFIG_BITS_PER_ROW-1:0] = {config_bit_tile_col_0, ..., config_bit_tile_col_last, config_bit_east_tile}
      · config_bit_tile_col_i = {sbox_0, ... sbox_last, cbox_x_in_0, ..., cbox_x_in_last,cbox_y_in_0, ..., cbox_y_in_last, cbox_x_out_0, ..., cbox_x_out_last, analog_0, ..., analog_last} corresponding to the tile in the ith column starting from the left
      · config_bit_east_tile = {sbox_0, ... sbox_last, cbox_y_0, ... c_box_y_last}
  - For the north row:
    * Response Length: NUM_CONFIG_BITS_PER_NORTH_ROW fill up to byte
    * Response Format:
      · response[NUM_CONFIG_BITS_PER_NORTH_ROW-1:0] = {config_bit_north_tile_col_0, ..., config_bit_north_tile_col_last, config_bit_northeast_tile}
      · config_bit_northtile_col_i = {sbox_0, ... sbox_last, cbox_x_in_0, ..., cbox_x_in_last} corresponding to the tile in the ith column starting from the left
      · config_bit_northeast_tile = {sbox_0, ... sbox_last}
  - Note: After reading config, the config enable would be turned off. Another enable command needs to be sent to re-enable the config enable.

6. **Enable Config Bit**:

- Send Command:

- Command Length: 8 bits
- Command Format:
  - command[FLD_CMD_TYPE] = CMD_ENABLE_CONFIG
  - command[FLD_ENABLE_DISABLE_CONFIG] = 1

- Receive Response:
  - Response Length: 8 bits
  - Response Format:
    - response[7:0] = 0. The response is used to indicate the operation is completed.

7. **Disable Config Bit**:

- Send Command:
  - Command Length: 8 bits
  - Command Format:
    - command[FLD_CMD_TYPE] = CMD_ENABLE_CONFIG
    - command[FLD_ENABLE_DISABLE_CONFIG] = 0
- Receive Response:
  - Response Length: 8 bits
  - Response Format:
    - response[7:0] = 0. The response is used to indicate the operation is completed.

8. **Reset All**:

- Send Command:
  - Command Length: 8 bits
  - Command Format:
    - command[FLD_CMD_TYPE] = CMD_RESET
- Receive Response:
  - Response Length: 8 bits
  - Response Format:
    - response[7:0] = 0. The response is used to indicate the operation is completed.

Listing 1: defines.v

```
// ------------------------------------------------------------
// GLOBAL DEFINITIONS
// ------------------------------------------------------------

// ======== UART_CLK_CYCLES_PER_BIT_DEFAULT ========
`define UART_CLK_CYCLES_PER_BIT_DEFAULT    260    // = 5000000/19200
`define W_UART_CLK_CYCLES_PER_BIT          9
`define BYTE_UART_CLK_CYCLES_PER_BIT       2
`define W_BYTE_UART_CLK_CYCLES_PER_BIT     2
`define NUM_BITS_TO_FIT_BYTE_FOR_UART_CLK_CYCLES_PER_BIT 16
```

```
// Oscillator Architectural Constants
`define NUM_ROWS              30
`define NUM_COLS              10
`define NUM_OSC_PER_ROW       9
`define NUM_OSC_PER_COL       29
`define NUM_OSC               261

`define BYTE_OSC_ONE_HOT      33
`define W_BYTE_OSC_ONE_HOT    6

`define W_NUM_COLS            4

// Frequency Calibration Constants
`define W_FREQ_COUNTER        20
`define W_FREQ_DIV            14
`define W_MUX_SEL             9
`define W_COUNT_NUM_COL       4

`define BYTE_FREQ_COUNTER     3
`define W_BYTE_FREQ_COUNTER   2

// ------------------------------------------------------------
// LOCAL OSCILLATOR COMMANDS
// ------------------------------------------------------------

// Config bits
`define NUM_CONFIG_BITS_PER_TILE                4422
`define NUM_CONFIG_BITS_PER_EAST_TILE           1200
`define NUM_CONFIG_BITS_PER_NORTH_TILE          1200
`define NUM_CONFIG_BITS_PER_EAST_NORTH_TILE     480
`define NUM_CONFIG_BITS_PER_ROW                 40998
`define NUM_CONFIG_BITS_PER_NORTH_ROW           11280

`define W_CONFIG_BYTE_COUNTER 13

`define BYTE_NUM_CONFIG_BITS_PER_ROW_ONE_HOT         5125
`define BYTE_NUM_CONFIG_BITS_PER_NORTH_ROW_ONE_HOT   1410

// ------------------------------------------------------------
// GLOBAL COMMANDS
// ------------------------------------------------------------

// Instruction Field Widths
`define W_CMD                 24
`define BYTE_CMD              3
`define W_CMD_TYPE            3
`define W_OSC_COL_ID          4
`define W_OSC_ROW_ID          5
`define W_FREQ_CAL            5
`define W_CONFIG_ROW_IDX      5
```

```
`define W_ENABLE_DISABLE_CONFIG   1

// Instruction Fields
`define FLD_CMD_TYPE          2:0
`define FLD_CONFIG_ROW_IDX    7:3
`define FLD_FRQ_CAL           7:3
`define FLD_OSC_COL_ID        11:8
`define FLD_OSC_ROW_ID        20:16
`define FLD_ENABLE_DISABLE_CONFIG  3 // 1: enable, 0: disable

// Command Types
`define CMD_NOP               `W_CMD_TYPE'b000
`define CMD_READ_FREQ         `W_CMD_TYPE'b001
`define CMD_READ_PHASE        `W_CMD_TYPE'b010
`define CMD_PRGM_CONFIG       `W_CMD_TYPE'b101
`define CMD_ENABLE_CONFIG     `W_CMD_TYPE'b100
`define CMD_READ_CONFIG       `W_CMD_TYPE'b110
`define CMD_RESET             `W_CMD_TYPE'b111

// UART Command/Response Bytes Counts
`define MAX_NUM_BITS_TO_FIT_BYTE_FOR_UART_DATA 264
`define W_UART_DATA_BTYE_COUNTER  13
`define W_RESPONSE 65536
```

D Testbench Example of BLIM Controller

This section provides examples of the testbench for the BLIM top-level controller. Due to the length of the original Verilog code, a simplified representation illustrating the test flow and structure is presented below.

Listing 2: Testbench Example

```
// ==== Initialization ====
init_testbench_environment();
// Initialize test environment and parameters
init_uart_clock_cycle_per_bit();
// Initialize UART clock cycles per bit in the controller

// ==== Test Frequency Reading ====
for (oscillator_index in num_oscillators)
    force_different_frequency_on_oscillator(oscillator_index);
    send_read_frequency_command(oscillator_index);
    check_response();
    // Compare the returned frequency value with the forced frequency

// ==== Test Phase Reading ====
while (iteration < max_iterations)
    force_random_phases_on_every_oscillator();
    send_read_phase_command();
    check_response();
```

```
    // Compare the returned phase value with the forced phase
    iteration++;

// ==== Test Configuration Bit Programming/Reading ====
generate_golden_data();
// Generate random values as the golden data for all configuration registers
for (tile_row_index in num_tile_rows)
    send_program_config_bit_command_on_tile_row(tile_row_index);
    check_config_bit_value_on_tile_row(tile_row_index);
    // Compare the register values in the tile row with the golden data
for (tile_row_index in num_tile_rows)
    send_read_config_bit_command_on_tile_row(tile_row_index);
    check_response();
    // Compare the returned config bit values with the golden data

// ==== Test Enable/Disable/Reset Configuration Bits ====
send_enable_command();
check_enable_in_tile_array();
// Verify that every enable signal in the tile array is on
send_disable_command();
check_disable_in_tile_array();
// Verify that every enable signal in the tile array is off
send_reset_command();
check_config_bit_value_on_tile_array();
// Verify that all register values in the tile array are zero
```