A Digital Oscillator-based Potts Machine for Graph Colouring Problems

Xingjian Gao and Jie Han

***Abstract*— Multi-state combinatorial optimization problems, such as graph colouring, are not naturally represented in binary Ising formulations. To address this issue, we propose a digital oscillator-based Potts machine (DOPM) implemented on a field programmable gate array (FPGA). This design translates the phase dynamics in the Potts machine into a discrete-time update rule suitable for implementations in digital circuits. Costly continuous functions and multiplications are approximated by using the signs of phase differences and a triangular-wave function for complex trigonometric functions, linear feedback shift registers for perturbation generation, and shift-and-add logic for coefficient scaling. To reduce the interconnect complexity, a processing element array is reused to update the graph row by row. Dual-buffered on-chip memories support row-wise processing between read and write banks, while a conflict monitor enables early stopping once a feasible colouring is detected. The DOPM finds zero-conflict 4-colourings for the King's graph with up to 7225 nodes. FPGA implementations show that it uses fewer hardware resources than state-of-the-art Ising machines and requires no digital signal processors for a problem of 2116 nodes.**

***Index Terms*— Potts Model, Oscillator-based Potts Machine, Combinatorial Optimization, Graph Colouring Problem, Field Programmable Gate Array**

I. Introduction

Combinatorial optimization problems (COPs) are often non-deterministic polynomial-time (NP)-hard or complete. They include frequency allocation in telecommunications [1], physical design in computer chips [2], and lithography layout decomposition [3]. As these problems scale up, conventional von Neumann architectures become increasingly inefficient [4]. To address these limitations, physics-inspired computing paradigms have emerged as promising alternatives, including Ising machines (IMs). By mapping the optimization target functions onto the Hamiltonian of a physical system, the natural evolution of an IM to the ground state provides a candidate solution for the optimization problem [5]. Several IMs using different technologies have been introduced in the literature [6]–[9].

While IMs have shown remarkable success in solving binary COPs (e.g., max-cut), their application to multi-state problems, such as graph colouring (GC), presents challenges. An IM uses q binary spins to represent a single variable with q possible states, e.g., using one-hot encoding, with additional constraints to ensure a valid state assignment [10].

The generalized form of the Ising model, the Potts model [11], is well-suited for solving multi-state problems, as its spins naturally possess q states. Several recent hardware implementations have explored this potential using optical [12], [13], hybrid (i.e., both optical and digital) [14], and complementary metal-oxide-semiconductor (CMOS) [15], [16] technologies. However, optical and hybrid designs are limited in scalability, robustness, and power efficiency.

This paper presents the design and implementation of a digital oscillator-based Potts machine (DOPM) optimized for field programmable gate array (FPGA) implementations. This design uses a discrete-time fixed-point implementation, eliminating the dependence on analog circuit dynamics and enabling a controlled digital implementation. To address the scalability challenge, we introduce a compact, folded architecture that reuses a processing element (PE) array across rows rather than assigning one PE per spin. This architecture uses a dual-buffered memory system to enable rapid row-parallel updates (e.g., 46 nodes per cycle), allowing the processing of large-scale problems (with 2116 nodes) on standard FPGA hardware while preserving the full eight-neighbour connectivity. Furthermore, the machine uses a cyclical annealing schedule to improve exploration in rugged energy landscapes. A low-overhead convergence monitor, implemented with OR logic, allows early stopping by identifying zero-conflict states during annealing, enabling an execution to end as soon as a valid colouring is found.

The DOPM is evaluated on King's graph problems, where nodes connected horizontally, vertically, or diagonally must be assigned with one of four different colours. Experimental results show that the DOPM implemented on FPGAs can find valid colouring for King's graph problems with 2116 nodes. In contrast, previous coupled CMOS ring-oscillator-based Potts implementations at similar scales reported accuracies below 100% [16], which indicates imperfect colourings. Additionally, simulations on 7225 nodes show that the architecture can still find conflict-free colouring in larger search spaces, although with a lower success rate.

The remainder of this paper is structured as follows. Section II outlines the theoretical foundations of the Potts model and oscillator-based Potts machines (OPMs). Section III details the proposed DOPM. Section IV presents experimental results on solving King's graph problems and hardware performance metrics. Finally, Section V concludes the paper.

II. Preliminaries

The Potts model [11] generalizes the Ising model [17] by describing the coupling of multi-state spins according to a graph topology. The classical Hamiltonian H_P for the Potts model is:

$$H_P = -\sum_{i,j} J_{ij}\delta(s_i, s_j), \tag{1}$$

where s_i or $s_j \in \{0, 1, 2, ..., q-1\}$ represents the spin state at the i^{th} or j^{th} node with q possible states, J_{ij} is the

The authors are with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada (e-mail: xg6@ualberta.ca; jhan8@ualberta.ca).

coupling strength between spins i and j, and $\delta(\cdot)$ is the Kronecker delta that equals 1 if $s_i = s_j$ or 0 otherwise.

The phase-based Potts model is the theoretical basis of OPMs, expressed as the vector Potts Hamiltonian [18]:

$$H_{vP} = -\sum_{i,j} J_{ij} \cos(\theta_{s_i} - \theta_{s_j}), \tag{2}$$

where each oscillator's phase θ_{s_i} effectively represents a discrete state s_i through the mapping $\theta_{s_i} = \frac{2\pi s_i}{q}$, where $s_i \in \{0, 1, ..., q-1\}$.

The OPM dynamics are formulated such that the evolution tends to minimize the energy in (2). Following the theoretical framework in [18], the oscillator phases are updated using the gradient descent on the energy function H_{vP}. The change rate of the i-th oscillator's phase is described by:

$$\begin{aligned} \frac{d\theta_i(t)}{dt} &= -\eta \left. \frac{\partial H_{vP}}{\partial \theta_i} \right|_{\boldsymbol{\theta}=\boldsymbol{\theta}(t)} \\ &= -\eta \sum_{j=1}^{N} J_{ij} \sin(\theta_i(t) - \theta_j(t)), \end{aligned} \tag{3}$$

where η is a positive scaling constant that determines the rate of phase evolution toward a lower-energy state, θ_i denotes the continuous phase of oscillator i.

The system described by (3) is mathematically equivalent to the Kuramoto model [18], [19]. To explore the energy landscape and enforce the state locking of the continuous phase θ_i to the discrete spin state θ_{s_i}, the formulation incorporates time-dependent annealing schedules along with a sub-harmonic injection locking (SHIL) term, following the framework in [18]:

$$\begin{aligned} \frac{d\theta_i}{dt} =& A_c(t) \sum_{j \in \mathcal{N}_i} J_{ij} \tanh\left(\beta \sin(\theta_i(t) - \theta_j(t))\right) \\ &- A_s(t) \sin(q\theta_i(t)) + A_n(t)\xi_i(t), \end{aligned} \tag{4}$$

where q represents the number of spin states and $\mathcal{N}_i$ denotes the set of nodes coupled to node i. The $\tanh(\beta\cdot)$ term adds a controllable nonlinearity to the coupling, with β controlling the steepness of the response to phase differences, thereby enhancing phase separation. The coefficients A_c, A_s, and A_n represent the coupling strength, the SHIL strength [18], and the noise amplitude, respectively. The SHIL term, $-A_s \sin(q\theta_i)$, locks oscillator phases to q quantized spin states.

III. A FULLY DIGITAL OSCILLATOR-BASED POTTS MACHINE

A. A Hardware-Oriented Algorithm Reformulation

To efficiently implement the continuous Kuramoto dynamics in (4) on an FPGA, the nonlinear terms must be approximated and mapped to low-cost digital logic. The following hardware-friendly approximations are developed to enable parallel processing while reducing hardware usage:

1) The Euler method is used to derive a discrete-time update rule for a fixed-point implementation. The continuous oscillator phase, $\theta_i \in [0, 2\pi)$, is encoded into a 16-bit unsigned integer, $\Theta_i \in [0, 2^{16})$. This mapping allows modulo-2π arithmetic to be performed naturally via discarding the carry-out of a 16-bit unsigned addition.
2) The model described in (4) uses zero-mean Gaussian noise for stochastic perturbation, which requires significant hardware resources. To mitigate it, we implemented 16-bit Fibonacci linear feedback shift registers (LFSRs) with the polynomial $x^{16} + x^{14} + x^{13} + x^{11} + 1$ as the noise source. Since the goal is combinatorial optimization rather than strict thermodynamic modelling, a finite-variance, zero-mean noise suffices for stochastic exploration. The LFSR output is centred by subtracting 0x8000, the midpoint in the range $[0, 2^{16})$, yielding bounded perturbations with much lower hardware requirements.
3) To avoid the high resource costs associated with trigonometric and exponential functions, and digital multipliers, we break down the continuous dynamics into logic-friendly operations:
   - Coupling Interaction (C_{ij}): The coupling interaction in (4) is defined as $C_{ij} = \tanh(\beta \sin(\Delta\theta))$, which involves complex trigonometric and nonlinear computations in hardware. When β is sufficiently large, this function tends to behave as a binary switch. For simplicity, we approximate it with the sign function, $C_{ij} = \mathrm{sgn}(\Delta\theta)$ and $C_{ij} \in \{-1, 0, 1\}$, which is effectively implemented using a single digital comparator.
   - Injection Locking (S): In (4), implementing the sinusoidal functions in the injection term $S = \sin(q\theta_i)$ on an FPGA is resource-heavy. Therefore, we approximate this term with a digital triangular waveform:

$$S = \mathrm{tri}(q\theta_i) = \begin{cases} \frac{2q\theta_i}{\pi}, & 0 \le q\theta_i < \frac{\pi}{2}, \\ 2 - \frac{2q\theta_i}{\pi}, & \frac{\pi}{2} \le q\theta_i < \frac{3\pi}{2}, \\ \frac{2q\theta_i}{\pi} - 4, & \frac{3\pi}{2} \le q\theta_i < 2\pi. \end{cases} \tag{5}$$

     This approximation maintains the multi-state locking behaviour and can be efficiently implemented using binary operators.
   - Coupling Scaling (F_i): For the King's graph, every valid edge has unit coupling, i.e., $J_{ij} = 1$, so J_{ij} is omitted in this work. In an 8-neighbour setup, the combined coupling term, $F_i = \sum_{j \in \mathcal{N}_i} C_{ij}$, is an integer between $[-8, 8]$. In the hardware implementation, A_c is stored as an 18-bit fixed-point number with 10 fractional bits, whereas A_s and A_n are stored as 4-bit unsigned integers for shifting. Utilizing this limited range, the multiplication with the dynamic coupling strength A_c is implemented without digital signal processors (DSPs). Let b_s be the sign bit of F_i, and given its magnitude in binary as $|F_i| = \sum_{k=0}^{3} b_k 2^k$, where $b_k \in \{0, 1\}$ is the k-th binary digit of $|F_i|$, the scaling is implemented using conditional shifts and additions:

$$A_c F_i = (-1)^{b_s} \sum_{k=0}^{3} b_k A_c 2^k. \tag{6}$$

     In hardware, the power-of-two factors 2^k are implemented as left shifts of A_c. Similarly, the injection

Algorithm 1 DOPM Cyclical Annealing Algorithm

Input: $\mathcal{N}$, F_{max}, C_a, and $F_{cyc} = F_{max}/C_a$
Output: $\mathbf{\Theta}_{opt}$

1: Initialize $\Theta_i \in [0, 2^{16} - 1]$ and LFSR seeds
2: **for** $f = 1$ to F_{max} **do**
3: **if** $f \bmod F_{cyc} == 1$ **then**
4: $A_c \Leftarrow A_{c,min}$; $A_s \Leftarrow 0$; $A_n \Leftarrow A_{n,max}$
5: **else**
6: $A_c \Leftarrow A_c + \delta_c$; $A_s \Leftarrow A_s + \delta_s$; $A_n \Leftarrow A_n - \delta_n$
7: **end if**
8: $conf\_global \Leftarrow 0$
9: **for** $r = 1$ to L **do**
10: $\mathbf{\Theta}_r[t] \Leftarrow$ Dual-Buffered Read Bank
11: **for** each node i in r **in parallel do**
12: $\Delta\Theta_i[t] \Leftarrow$ Phase increment from (7)
13: $\Theta_i[t+1] \Leftarrow \Theta_i[t] + \Delta\Theta_i[t]$
14: $conf\_local_i \Leftarrow Check(\Theta_i[t+1], \mathcal{N}_i)$
15: **end for**
16: Dual-Buffered Write Bank $\Leftarrow \mathbf{\Theta}_r[t+1]$
17: $conf\_global \Leftarrow conf\_global \vee (\bigvee_i conf\_local_i) \vee CrossCheck(r, r+1)$
18: **end for**
19: **if** $conf\_global == 0$ **then**
20: **break**
21: **end if**
22: Swap Dual-Buffered Memory Banks
23: **end for**
24: $\mathbf{\Theta}_{opt} \Leftarrow \mathbf{\Theta}[t+1]$

and noise amplitudes are scaled via left shifts, thereby avoiding the use of DSP multipliers.

Built upon the earlier approximations, the phase increment for each PE in discrete time is given by:

$$\Delta\Theta_i[t] = K\Big(A_c[t]F_{c,i}[t] - S_i[t]2^{A_s[t]} + \xi_i[t]2^{A_n[t]}\Big). \quad (7)$$

The multiplications by $2^{A_s[t]}$ and $2^{A_n[t]}$ are implemented as left shifts, since $A_s[t]$ and $A_n[t]$ are unsigned integers. The constant K in (7) is the overall scaling factor in the fixed-point implementation of the Euler update, defined as:

$$K = C_{\mathrm{s}} \cdot 2^{-(S_{dt}+S_{qc})}, \quad (8)$$

where $C_{\mathrm{s}} = 10430 \approx 2^{16}/2\pi$ is the radian mapped to a 16-bit phase scaling factor. The parameter $S_{dt} = 9$ represents the Euler integration step, implemented as a right shift, corresponding to a discrete-time interval of roughly $\Delta t \approx 2^{-9}$, which controls the oscillator's rate of change. The value $S_{qc} = 10$ corresponds to a right shift that removes the 10 fractional bits added to the fixed-point format during scaling, returning the result to an integer. Multiplication by C_{s} is implemented using a shift-and-add network to avoid using DSP blocks. The oscillator phase is then updated using the Euler method as

$$\Theta_i[t+1] = \Theta_i[t] + \Delta\Theta_i[t]. \quad (9)$$

Since Θ_i in the interval $[0, 2\pi)$ is stored as a 16-bit unsigned integer, retaining only the lower 16 bits after addition, directly implements the periodicity of $[0, 2\pi)$ without explicit modulo operations.

B. A Discrete-Time Update Algorithm

In this work, the phase increment equation (7) is iteratively applied over four annealing cycles. As shown in Algorithm 1, the DOPM requires external input data for the graph adjacency information, the maximum number of frames, F_{max}, where one frame denotes a complete row-wise sweep over the graph, and the total number of annealing cycles, C_a. The output is the optimized phase configuration matrix, $\mathbf{\Theta}_{opt}$, from which discrete graph colours are directly obtained using the most significant bits (MSBs).

Initially, the oscillator phases and the seeds for LFSR-generated noise are assigned using uniform random numbers (Line 1). At the start of each annealing cycle, the annealing parameters are reset to encourage exploration of the solution space. The coupling strength A_c is set to its lowest value, the injection shift A_s is cleared, and the noise shift A_n is maximized to boost stochastic perturbation. During each of the cycle, A_c is increased by δ_c to gradually penalize monochromatic edges, while A_s and A_n are updated by δ_s and δ_n, respectively, to improve state locking and decrease stochastic noise (Lines 3–7). Before the row-wise evaluation begins, the global conflict flag, $conf\_global$, is reset to zero (Line 8).

To reduce the resource usage for routing, the graph is processed row by row across all L rows, with L denoting the number of rows in the King's graph. For each rth row, the current phase values, $\mathbf{\Theta}_r[t]$, are fetched from the read memory bank (Line 10). Then, all nodes in the row compute the phase increment according to (7) (Line 12), before being updated for the next time step via 16-bit unsigned addition (Line 13). A parallel constraint check is performed on the MSBs of the updated phases to detect colouring violations, which generates local conflict indicators, $conf\_local_i$ (Line 14). This row is then updated in the write bank, while local conflicts and a cross-check against the next unupdated row are simultaneously taken into account for the frame-level flag, $conf\_global$, through an OR-reduction, implemented as a tree of OR gates that sets $conf\_global$ to 1 if any local conflict indicator is high (Lines 16-17).

Using separate read and write banks avoids read-after-write hazards during streaming updates. After processing all rows, a finite-state machine (FSM) checks $conf\_global$. If no conflict is detected, a valid colouring has been obtained, and the execution terminates (Lines 19-21). Otherwise, the read and write banks are swapped, and the next frame begins (Line 22).

C. Circuit Design

Algorithm 1 employs a folded row-parallel architecture, reusing an array of PEs to update the graph one row at a time. This setup avoids creating a PE for each node and simplifies the global interconnect. The hardware comprises three components: a top-level controller, dual-buffered memories, and a Row Engine with parallel PEs. The Row Engine is named after its role in row-based processing,

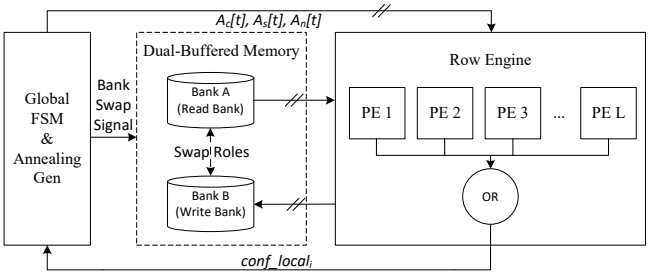

Fig. 1. The top-level diagram of DOPM.

which generates the three-row neighbourhood window and schedules PE operations.

1) Top-Level Control and Dual-Buffered Memories: As illustrated in Fig. 1, a global FSM generates annealing parameters and coordinates frame execution. The design creates $L = 46$ identical processing columns, each linked to a dual-buffered memory bank. In each frame, one bank acts as the read bank to stream the current row phase values, while the other serves as the write bank to store updated phases. Once all rows in the frame are processed, the roles of the two banks are swapped (Line 22).

2) Row Engine Microarchitecture: The Row Engine in Fig. 2 coordinates the operation of the parallel PEs and provides information about neighbouring spatial elements. It uses a three-row sliding window, implemented as a register chain (top_i, mid_i, bot_i). For boundary rows, the system detects out-of-range indices and masks them through multiplexers, effectively zero-padding invalid neighbours without changing the datapath in a PE. An OR-reduction tree combines conflict flags from all PEs into a row-level conflict signal. The top-level controller then uses this signal to update $conf\_global$. Thus, $conf\_global$ is set to 1 if any PE detects a colouring conflict in the current frame.

3) Pipelined PE Datapath: Each PE assesses the specific update rule through a four-stage pipeline consisting of subtractors, adders, shifters, and comparators, as shown in Fig. 3:

- Stage 0 (Feature Extraction): Phase differences with valid neighbours are calculated simultaneously. The sign of each phase difference is determined and summed to produce the discrete coupling term. At the same time, the triangular injection term $\mathrm{tri}(q\Theta_i)$ and a perturbation noise centred around zero from a 16-bit LFSR are generated in parallel.
- Stage 1 (Term Scaling & Accumulator): The annealing parameters A_c, A_s, and A_n scale the coupling, injection, and noise terms using programmable shift-and-add logic. Intermediate sums are accumulated within a fixed 28-bit datapath to avoid unnecessary bit-width increase.
- Stage 2 (Integration): The weighted terms are summed and then arithmetically right-shifted by S_{dt}, preserving the sign while implementing the discrete Euler step to generate the intermediate increment $\Delta\Theta_i$.
- Stage 3 (Final Scaling & Conflict Check): The increment $\Delta\Theta_i$ is multiplied by C_s using a truncated shift-and-add structure limited to 26 bits. The result is then right-shifted by S_{qc} bits and added to the current phase, with rounding handled via 16-bit unsigned overflow. The MSBs are then compared with neighbouring colour bits using comparators. Additional logic is used to mask invalid neighbours, producing $conf\_local_i$.

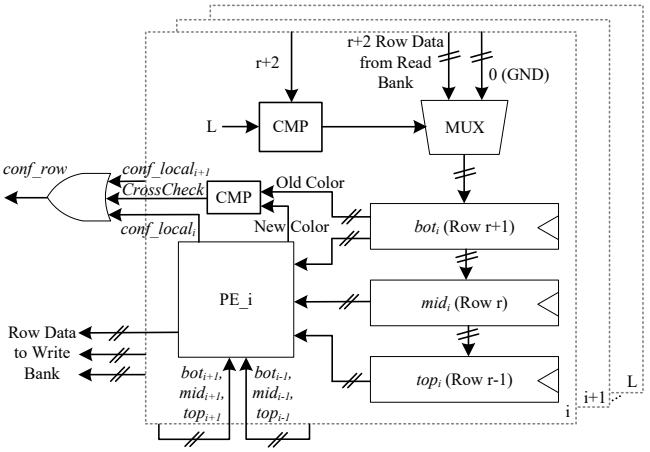

Fig. 2. Row Engine microarchitecture with a three-row sliding window, boundary masking, and OR-based conflict detection.

D. Timing Design

As shown in Fig. 4, the Row Engine functions as a 6-phase streaming controller (Phases A–F), coordinating continuous block random access memory (BRAM) access, the 4-stage pipelined datapath in a PE, and row-wise neighbourhood reuse. Under this schedule, processing one row takes six clock cycles.

1) Phase A (Launch & Pre-fetch): The controller sets the read address for row r and simultaneously begins pre-fetching row $r+2$ to maintain the three-row sliding window in future iterations. Meanwhile, the LFSRs advance to generate the noise, and the PE array starts Stage 0 with the data in the current window.
2) Phases B & C (Pipeline Advance): Intermediate signals progress through Stage 1 and Stage 2 of the pipeline registers on each rising edge. During these steps, the PE performs the shift-and-add scaling operations and computes the intermediate increment $\Delta\Theta_q$.
3) Phase D (Registering Updated Values): Stage 3 finalizes the ongoing scaling and phase update. The updated row phases, along with their corresponding per-column conflict flags, are stored in Row Engine registers before being written to the write bank.
4) Phase E (Write-back & Window Shift): The modified row $\mathbf{\Theta}_r[t+1]$ is stored in the dual-buffered write bank. The sliding window registers are updated ($top_i \leftarrow mid_i$, $mid_i \leftarrow bot_i$), and the prefetched row $r+2$ is loaded into bot_i.
5) Phase F (Global Reduction & Optional Copy): The OR-tree simplifies local conflict flags to update $conf\_global$. If a zero-conflict setup is detected before all rows in the current frame have been processed, the controller can switch to an early-stop copy mode: it directly copies the remaining rows from the read bank to the write bank, skipping the PE pipeline. This essential feature keeps data consistency across dual-buffered memories for future evaluations while avoiding unnecessary PE activities after a feasible solution is found.

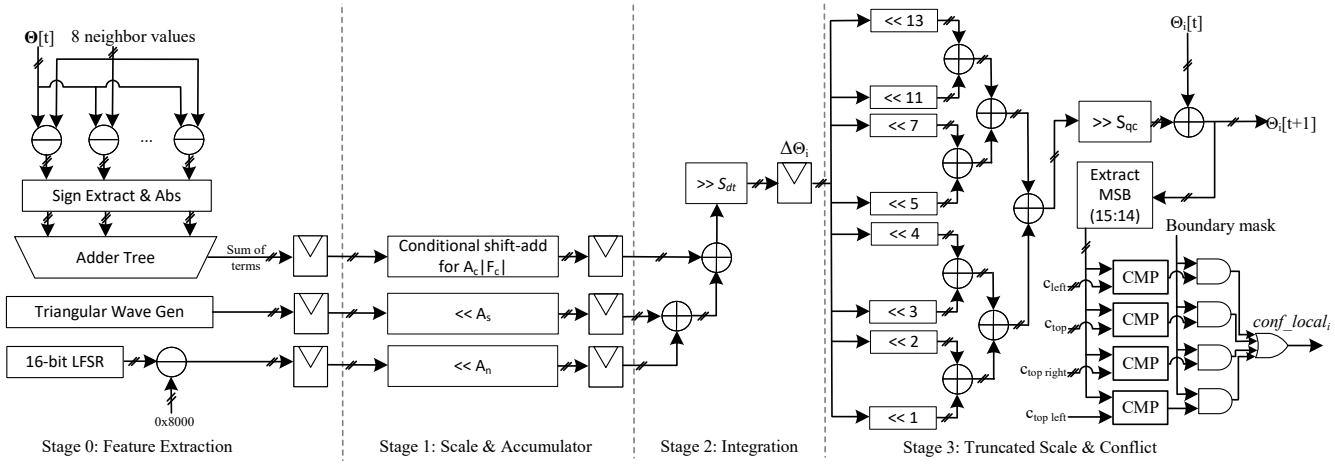

Fig. 3. The PE's internal datapath includes a 4-stage pipeline and an entirely DSP-free arithmetic network.

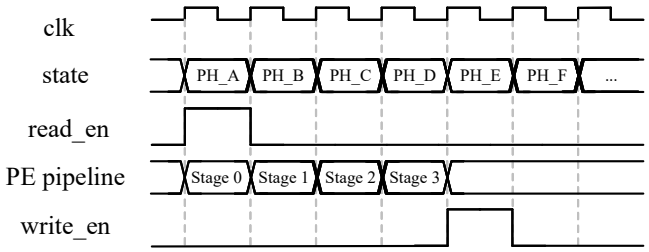

Fig. 4. The timing design of the DOPM.

IV. Experimental Results

A. Solving King's Graph Problem

To evaluate the proposed discrete-time update rule independently of hardware implementations, we first run it on an Intel Core Ultra 7 265F processor using Python 3.13.2. The software implements the same hardware-inspired formulation, including a 16-bit unsigned phase representation, ternary sign-based coupling, a triangular-wave approximation of the injection term, and centred perturbations by using 16-bit LFSRs. Consequently, the results reflect the discretized digital dynamics designed for FPGA implementations, rather than that of the ideal continuous-time OPM model.

We adopt a strict criterion: a run is considered successful only if the final colouring has no conflicts, meaning no edge connects two nodes of the same colour. The zero-conflict success rate is the percentage of successful runs out of 100 independent trials, each starting with a different random initial phase and noise seed.

Optical Potts-machine implementations [12], [14] have primarily been demonstrated on small graphs with fewer than 50 nodes. Coupled CMOS ring oscillator-based Potts machines [15], [16] have been tested at larger scales, reaching approximately 2000 nodes, with performance typically measured by node-wise accuracy, i.e., the fraction of nodes assigned the correct colour. However, since any accuracy below 100% indicates that at least one constraint is violated, those results do not necessarily imply conflict-free colouring.

In contrast, the DOPM design is evaluated by following the zero-conflict criterion. As shown in Table I, the success rate decreases with an increasing problem size, but conflict-free solutions are still achieved at all scales tested, up to

TABLE I

Performance and Scaling of Potts Machines solving GCPs

Design	GCP	Scale	Evaluated Metric	Result
DOPM	4-coloring	49	Success Rate	99%
DOPM	4-coloring	400	Success Rate	66%
DOPM	4-coloring	1,024	Success Rate	61%
DOPM	4-coloring	2,116	Success Rate	18%
DOPM	4-coloring	7,225	Success Rate	4%
Optical [12]	3-coloring	30	Accuracy	50% - 100%
Hybrid [14]	4-coloring	47	Success Rate	50%
CMOS [15]	3-coloring	2,000	Accuracy	83% - 92%[1]
CMOS [16]	4-coloring	2,116	Accuracy	96% - 97%[1]

[1] Node accuracy below 100% indicates that at least one colouring constraint has been violated.

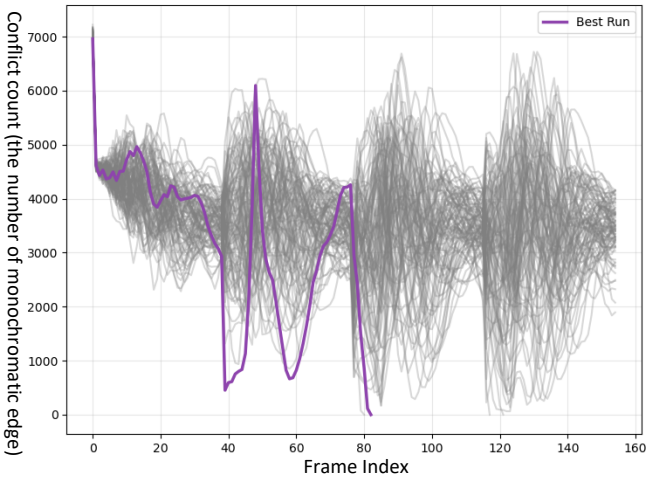

Fig. 5. Conflict count versus the frame index for the King's graph problem with $N = 7225$ over 100 runs. The darker curve shows one of the best runs, which reaches zero conflicts.

$N = 85 \times 85 = 7225$, as shown in Fig. 5. This indicates that the proposed digital update rule continues to identify feasible solutions even as the problem size increases. Repeating independent trials increases the chance of finding a correct solution. The probability of obtaining at least one feasible solution after k independent runs is $1 - (1 - \hat{p})^k$, where $\hat{p}$ is the empirical success rate. For example, for $N = 2116$, $\hat{p} = 0.18$, the success probability over 1000 independent runs is $1 - 0.82^{1000} > 0.99$.

TABLE II
FPGA IMPLEMENTATIONS OF VARIOUS HARDWARE SOLVERS

Design	Target Problem	Size	LUTs	FFs	BRAMs	DSPs
DOPM	King's Graph	2,116	33,035	12,492	46	0
SOIM [7]	Max-Cut	500	58,989	34,631	-	500
FA [9]	Max-Cut	1,024	468,183	-	-	-
SBM [8]	Max-Cut	2,048	260,953	281,274	1012	104

B. The Hardware Performance

Table II presents the post-synthesis resource usage of the proposed DOPM on a Xilinx Kintex-7 XC7K160T (Vivado 2025.1), along with representative FPGA-based Ising solvers from prior studies. Note that the simulated oscillator-based Ising machine (SOIM) [7], the FPGA annealer (FA) [9], and the simulated bifurcation Machine (SBM) [8] are designed for binary COPs based on Ising models, such as max-cut, while DOPM targets a multi-state Potts model for GC, updating four-state variables instead of binary spins. Therefore, Table II should be interpreted only as a reference rather than a direct comparison.

Although targeting a multi-state Potts formulation with $N = 2116$, the DOPM requires a low hardware resource usage. Notably, the design does not use any DSP blocks. Unlike fully parallel architectures such as the SOIM, where DSP usage scales with the network size due to multiplier-based local-field calculations, the DOPM employs shift-and-add for all arithmetic operations, eliminating the need for dedicated DSP blocks.

The folded row-parallel organization further reduces logic and memory needs by reusing a PE array for a row-by-row processing of the King's graph. The 46×46 King's graph is implemented using 46 on-chip BRAMs, with each PE connected to a dual-buffered memory bank. The entire design uses 33,035 look-up tables (LUTs) and 12,492 flip-flops (FFs). Compared to the FA and SBM, this architecture handles larger problems with significantly less hardware usage, as shown in Table II. These findings indicate that the DOPM provides a hardware-efficient digital implementation for large-scale multi-state combinatorial optimization on FPGA systems.

V. CONCLUSIONS

This paper details the design and FPGA implementation of the DOPM for multi-state combinatorial optimization. The continuous-time Kuramoto–Potts dynamics are adapted into a discrete-time digital update rule compatible with fixed-point arithmetic and FPGA implementations. This method avoids dependence on the behaviour of analog devices, ensuring robust digital operation.

Under a strict zero-conflict criterion, this design produces conflict-free colouring at a scale comparable to that of a previous coupled CMOS ring oscillator-based Potts machine, which reported less than 100% accuracy. Additionally, the DOPM can find conflict-free solutions on King's Graph with up to 7,225 nodes. This suggests that the architecture retains the ability to explore complex energy landscapes and avoid suboptimal local minima.

From a hardware design perspective, the folded row-parallel architecture simplifies the global interconnect by reusing a PE array for row-by-row processing. Arithmetic operations are performed using shift-and-add structures, removing the need for DSPs.

Overall, the DOPM provides a scalable digital design framework for multi-state Potts-based optimization on FPGA platforms. Future research will examine support for weighted graphs and integration into heterogeneous computing systems.

REFERENCES

[1] W. K. Hale, "Frequency assignment: Theory and applications," *Proceedings of the IEEE*, vol. 68, no. 12, pp. 1497–1514, 1980.

[2] S. Held, B. Korte, D. Rautenbach, and J. Vygen, "Combinatorial optimization in VLSI design," *Combinatorial Optimization*, pp. 33–96, 2011.

[3] J. Kuang and E. F. Young, "An efficient layout decomposition approach for triple patterning lithography," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 1–6.

[4] I. L. Markov, "Limits on fundamental limits to computation," *Nature*, vol. 512, no. 7513, pp. 147–154, 2014.

[5] T. Zhang, Q. Tao, B. Liu, A. Grimaldi, E. Raimondo, M. Jiménez, M. J. Avedillo, J. Nunez, B. Linares-Barranco, T. Serrano-Gotarredona *et al.*, "A review of Ising machines implemented in conventional and emerging technologies," *IEEE Transactions on Nanotechnology*, vol. 23, pp. 704–717, 2024.

[6] T. Wang and J. Roychowdhury, "Oscillator-based Ising machine," *arXiv preprint arXiv:1709.08102*, 2017.

[7] B. Liu, T. Zhang, X. Gao, and J. Han, "An efficient simulated oscillator-based Ising machine on FPGAs," in *the 24th International Conference on Nanotechnology (NANO)*. IEEE, 2024, pp. 469–474.

[8] K. Tatsumura, A. R. Dixon, and H. Goto, "FPGA-based simulated bifurcation machine," in *the 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 59–66.

[9] C. Yoshimura, M. Hayashi, T. Okuyama, and M. Yamaoka, "FPGA-based annealing processor for Ising model," in *the Fourth International Symposium on Computing and Networking (CANDAR)*. IEEE, 2016, pp. 436–442.

[10] A. Lucas, "Ising formulations of many NP problems," *Frontiers in physics*, vol. 2, p. 5, 2014.

[11] F.-Y. Wu, "The Potts model," *Reviews of modern physics*, vol. 54, no. 1, p. 235, 1982.

[12] M. Honari-Latifpour and M.-A. Miri, "Optical Potts machine through networks of three-photon down-conversion oscillators," *Nanophotonics*, vol. 9, no. 13, pp. 4199–4205, 2020.

[13] K. Inoue, K. Yoshida, and S. Kitahara, "Coherent Potts machine based on an optical loop with a multilevel phase-sensitive amplifier," *Optics Communications*, vol. 528, p. 129022, 2023.

[14] K. Inaba, T. Inagaki, K. Igarashi, S. Utsunomiya, T. Honjo, T. Ikuta, K. Enbutsu, T. Umeki, R. Kasahara, K. Inoue *et al.*, "Potts model solver based on hybrid physical and digital architecture," *Communications Physics*, vol. 5, no. 1, p. 137, 2022.

[15] Y. E. Gonul and B. Taskin, "Multi-Phase coupled CMOS Ring Oscillator based Potts Machine," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–9.

[16] ——, "A Multi-Stage Potts Machine based on Coupled CMOS Ring Oscillators," in *Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2025, pp. 1–7.

[17] E. Ising, "Beitrag zur theorie des ferromagnetismus," *Zeitschrift für Physik*, vol. 31, no. 1, pp. 253–258, 1925.

[18] J. Roychowdhury and S. Seal, "Oscillator-based Potts machine (OPM) for the implementation of the vector Potts model," *EECS University of California, Berkeley*, 2022.

[19] J. A. Acebrón, L. L. Bonilla, C. J. Pérez Vicente, F. Ritort, and R. Spigler, "The Kuramoto model: A simple paradigm for synchronization phenomena," *Reviews of modern physics*, vol. 77, no. 1, pp. 137–185, 2005.