**Title**
Novel Computing Paradigms using Oscillators



**Author**
Wang, Tianshi

**Publication Date**
2019

Peer reviewed|Thesis/dissertation

# Novel Computing Paradigms using Oscillators

by

Tianshi Wang

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering – Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Jaijeet Roychowdhury, Chair
Professor Kristofer Pister
Professor Bruno Olshausen

Fall 2019

Novel Computing Paradigms using Oscillators



Tianshi Wang

# Abstract

Novel Computing Paradigms using Oscillators

by

Tianshi Wang

Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Jaijeet Roychowdhury, Chair

This dissertation is concerned with new ways of using oscillators to perform computational tasks. Specifically, it introduces methods for building finite state machines (for general-purpose Boolean computation) as well as Ising machines (for solving combinatorial optimization problems) using coupled oscillator networks.

But firstly, why oscillators? Why use them for computation?

An important reason is simply that oscillators are fascinating. Coupled oscillator systems often display intriguing synchronization phenomena where spontaneous patterns arise. From the synchronous flashing of fireflies to Huygens' clocks ticking in unison, from the molecular mechanism of circadian rhythms to the phase patterns in oscillatory neural circuits, the observation and study of synchronization in coupled oscillators has a long and rich history. Engineers across many disciplines have also taken inspiration from these phenomena, *e.g.*, to design high-performance radio frequency communication circuits and optical lasers. To be able to contribute to the study of coupled oscillators and leverage them in novel paradigms of computing is without question an interesting and fulfilling quest in and of itself.

Moreover, as Moore's Law nears its limits, new computing paradigms that are different from mere conventional complementary metal–oxide–semiconductor (CMOS) scaling have become an important area of exploration. One broad direction aims to improve CMOS performance using device technology such as fin field-effect transistors (FinFET) and gate-all-around (GAA) FETs. Other new computing schemes are based on non-CMOS material and device technology, *e.g.*, graphene, carbon nanotubes, memristive devices, optical devices, *etc.*. Another growing trend in both academia and industry is to build digital application-specific integrated circuits (ASIC) suitable for speeding up certain computational tasks, often leveraging the parallel nature of unconventional non-von Neumann architectures. These schemes seek to circumvent the limitations posed at the device level through innovations at the system/architecture level.

Our work on oscillator-based computation represents a direction that is different from the above and features several points of novelty and attractiveness. Firstly, it makes meaningful use of nonlinear dynamical phenomena to tackle well-defined computational tasks that span analog and digital domains. It also differs from conventional computational systems at the fundamental logic encoding level, using timing/phase of oscillation as opposed to voltage levels to represent logic values. These differences bring about several advantages. The change of logic encoding scheme has several device- and system-level benefits related to noise immunity and interference resistance. The use of nonlinear oscillator dynamics allows our systems to address problems difficult for conventional digital computation. Furthermore, our schemes are amenable to realizations using almost all types of oscillators, allowing a wide variety of devices from multiple physical domains to serve as the substrate for computing. This ability to leverage emerging multiphysics devices need not put off the realization of our ideas far into the future. Instead, implementations using well-established circuit technology are already both practical and attractive.

This work also differs from all past work on oscillator-based computing, which mostly focuses on specialized image preprocessing tasks, such as edge detection, image segmentation and pattern recognition. Perhaps its most unique feature is that our systems use transitions between analog and digital modes of operation — unlike other existing schemes that simply couple oscillators and let their phases settle to a continuum of values, we use a special type of injection locking to make each oscillator settle to one of the several well-defined multistable phase-locked states, which we use to encode logic values for computation. Our schemes of oscillator-based Boolean and Ising computation are built upon this digitization of phase; they expand the scope of oscillator-based computing significantly.

Our ideas are built on years of past research in the modelling, simulation and analysis of oscillators. While there is a considerable amount of literature (arguably since Christiaan Huygens wrote about his observation of synchronized pendulum clocks in the 17th century) analyzing the synchronization phenomenon from different perspectives at different levels, we have been able to further develop the theory of injection locking, connecting the dots to find a path of analysis that starts from the low-level differential equations of individual oscillators and arrives at phase-based models and energy landscapes of coupled oscillator systems. This theoretical scaffolding is able not only to explain the operation of oscillator-based systems, but also to serve as the basis for simulation and design tools. Building on this, we explore the practical design of our proposed systems, demonstrate working prototypes, as well as develop the techniques, tools and methodologies essential for the process.

To Lu Wang.

# Contents

**Contents** ii

**List of Figures** iv

**List of Tables** x

**1 Introduction** 1

1.1 Computing beyond Moore's Law . . . . . 1

1.2 Oscillator-based Computation . . . . . 3

1.3 Overview of this Dissertation . . . . . 4

**2 Injection Locking in Oscillators: Theory and Analysis** 6

2.1 Phase Macromodels of Oscillators . . . . . 6

2.2 Injection Locking and the Generalized Adler's Equation . . . . . 7

2.3 Models of Coupled Oscillators . . . . . 11

**3 Boolean Computation using Oscillators** 15

3.1 History of Phase-encoded Computation . . . . . 16

3.2 PHLOGON: PHase-based LOGic using Oscillatory Nano-systems . . . . . 20

3.2.1 Oscillator Latches with Phase Logic . . . . . 20

3.2.2 General-purpose Computing with Phase Logic . . . . . 23

3.2.3 Differences from Previous Work . . . . . 26

3.3 System Designs and Prototypes . . . . . 28

3.3.1 PHLOGON Systems with CMOS Ring Oscillators . . . . . 29

3.3.2 PHLOGON Systems with CMOS LC Oscillators . . . . . 37

3.3.3 Achieving SHIL and Bit Storage in Mechanical Metronomes . . . . 39
3.4 Discussion on Oscillator-based Boolean Computation . . . . . 51
3.4.1 Inherent Noise Immunity of Phase-based Logic . . . . . 51
3.4.2 Potential Power/energy Advantages of PHLOGON . . . . . 53
3.4.3 Power-speed Trade-offs in PHLOGON Systems . . . . . 54

**4 Oscillator-based Ising Machines 59**
4.1 The Ising problem and Existing Ising Machine Approaches . . . . . 61
4.2 Mechanism of Oscillator-based Ising Machines . . . . . 63
4.2.1 Global Lyapunov Function of Coupled Oscillators . . . . . 64
4.2.2 Coupled Oscillators under SHIL and the Global Lyapunov Function 65
4.2.3 Stochastic Model of Oscillator-based Ising Machines . . . . . 67
4.2.4 Coupled Oscillator Networks with Frequency Variations . . . . . 68
4.3 Examples . . . . . 69
4.3.1 Small MAX-CUT Problems . . . . . 69
4.3.2 MAX-CUT Benchmark Problems . . . . . 73
4.3.3 A Graph Coloring Example . . . . . 78
4.3.4 Hardware Prototypes . . . . . 79

**5 Design Tools: Berkeley Model and Algorithm Prototyping Platform 83**
5.1 Existing Open-source Simulators and Motivation for MAPP . . . . . 84
5.2 Features of MAPP . . . . . 85
5.3 Compact Modelling in MAPP . . . . . 87
5.3.1 Modelling Nonlinear Devices with Hysteresis . . . . . 94
5.3.2 Modelling Multiphysics Devices and Systems . . . . . 110
5.4 Prototyping Simulation Algorithms in MAPP . . . . . 114
5.5 Using MAPP for Oscillator-based Computation . . . . . 116

**6 Conclusions and Future Work 120**
6.1 Conclusions . . . . . 120
6.2 Future Work . . . . . 121

# List of Figures


3.1 von Neumann's basic phase-based latch: a nonlinear AC-pumped circuit with multiple subharmonic steady states. . . . . 16

3.2 von Neumann's scheme for setting a latch to an input state and retaining it. . 17

3.3 von Neumann's scheme for combinatorial logic in phase. . . . . 18

3.4 Goto's design of parametrons and parametron-based computing systems. . . 19

3.5 Illustration of frequency and phase lock of injection locking. . . . . 21

3.6 Illustration of frequency and phase lock of sub-harmonic injection locking. 21

3.7 GAE equilibrium equation (3.1) establishes multistability and input phase acquisition properties of oscillator latches. . . . . 22

3.8 A general FSM. SYNC is used to develop multi-stability for encoding phase logic; NOT/MAJORITY gates are for combinational logic operations. . . . 23

3.9 Phase-domain plots illustrating NOT and MAJORITY operations. . . . . 24

3.10 Design and implementation of phase-based D latch. . . . . 24

3.11 Serial adder. . . . . 25

3.12 State transition diagram of a serial adder. . . . . 25

3.13 Waveforms from adding $a = b = 101$ with serial adder in Fig. 3.11 implemented using ring oscillators. . . . . 26

3.14 Circuit schematic of a 3-stage ring oscillator. . . . . 29

3.15 Photo of a 3-stage ring oscillator on breadboard. . . . . 29

3.16 Results seen from an oscilloscope. . . . . 29

3.17 Simulation results from ngspice and MAPP. . . . . 29

3.18 Diagram of D latch made from a 3-stage ring oscillator. . . . . 30

3.19 Transient responses of the 3-stage ring oscillator with different SYNC magnitudes: $20\mu$A, $50\mu$A, $100\mu$A. . . . . 30

3.20 First entry $\vec{v}_1(t)$ of the PPV of the 3-stage ring oscillator with SYNC. . . . 31

3.21 Transient simulation results showing the operation of the D latch in Fig. 3.18. 32

3.22 Breadboard implementation of the D latch in Fig. 3.18. (a) breadboard photo; (b) waveforms showing the two logic states (yellow) in the D latch when EN=0. REF is blue; SYNC is pink. . . . . 33

3.23 Diagrams of phase-based logic latches. . . . . 33

3.24 Circuit schematic of the phase logic latch in Fig. 3.23(a). . . . . 34

3.25 Breadboard implementation of the phase logic latch in Fig. 3.24. . . . . 34

3.26 Entries of PPV for SYNC and SIG1 inputs of the oscillator latch in Fig. 3.24. 35

3.27 Test results of the phase logic D latch seen from an oscilloscope. . . . . 35

3.28 Breadboard implementation of the serial adder in Fig. 3.11. The green blocks highlight the flip-flop made from two D latches; the yellow block is the combinational logic block that has the functionality of a full adder. . . 36

3.29 Test results of the master-slave D flip-flop as shown in the green block of Fig. 3.28. REF, output of master, output of slave are displayed with blue, green, yellow respectively. . . . . 37

3.30 Selected test results of the serial adder as in Fig. 3.28. REF, sum, cout are displayed with blue, green, yellow respectively. . . . . 37

3.31 Schematic of a complementary cross-coupled LC oscillator. . . . . 38

3.32 PPV entry of the cross-coupled LC oscillator for the SYNC input. . . . . 38

3.33 Comparison between waveforms from ring-oscillator- and LC-oscillator-based PHLOGON systems. SYNC is in pink; REF is in blue; signals from oscillator latches are in yellow. . . . . 38

3.34 Photo of breadboard circuit of an LC-oscillator-based PHLOGON system. 39

3.35 Metronomes standing on a rolling board end up ticking in unison. . . . . 40

3.36 Transient simulation results of $\theta$ and $\dot{\theta}$. . . . . 43

3.37 Transient simulation results of $g(\theta,\ \dot{\theta})$. . . . . 43

3.38 Input PPV of metronome. . . . . 44

3.39 Frequency domain coefficients of metronome's PPV. . . . . 44

3.40 Phase portrait of a metronome's oscillation before and after modification. . 44

3.41 Input PPV of modified metronome. . . . . 45

3.42 Frequency domain coefficients of modified metronome's PPV. . . . . 45

3.43 (a) Two metronomes placed on a stationary board; (b) corresponding waveforms of the tips of their pendulum rods; (c) two metronomes on a rolling board; (d) their corresponding waveforms. . . . . 46

3.44 (a) Lissajous curves Fig. 3.43 (b), showing that no synchronization happens; (b) Lissajous curves Fig. 3.43 (d), with the pattern demonstrating SHIL. . . 47

3.45 Phase difference between the 2Hz and 1Hz metronomes. The difference is measured by the number of cycles of the 1Hz metronome. Positive values mean that the 1Hz metronome is leading the 2Hz one. . . . . 47

3.46 Illustration of the experimental setup, where two 1Hz metronomes (with red and green tapes) oscillate in perpendicular directions, and one 2Hz metronome is placed between them with a 45° angle. . . . . 48

3.47 Photo of the experimental setup. . . . . 48

3.48 X or Y coordinates of the tips of the three metronomes in Fig. 3.47 during the experiment. . . . . 49

3.49 Excerpts from Fig. 3.48 and their corresponding Lissajous curves. . . . . 50

3.50 Mechanisms enhancing the noise resistance of phase-encoded logic. . . . . 51

3.51 Illustration of the definition of $Q$ factor for nonlinear oscillators. . . . . 55

3.52 A simple negative-resistance LC oscillator. Different choice of f(v) will result in different $Q$ factor of the oscillator. . . . . 56

3.53 Simulation results of amplitude (a1-3) and phase shifting (b1-3) of high-$Q$ and low-$Q$ oscillators. In (b3), we plot the zero-crossing differences between the v signals and injection signals in (b1) and (b2). . . . . 56

3.54 A 3-input averager used as MAJORITY gate in phase-encoded logic computation. . . . . 57

4.1 Schematic for size-4 OPO-based Ising machine (Fig. 2 in [1]). . . . . 62

4.2 Illustration of the basic mechanism of oscillator-based Ising machines: (a) oscillator shifts its natural frequency from $f_0$ to $f_1$ under external perturbation; (b) oscillator's phase becomes stably locked to the perturbation; (c) when the perturbation is at $2f_1$, the oscillator locks to its subharmonic at $f_1$; (d) bistable phase locks under subharmonic injection locking; (e) coupled subharmonically injection-locked oscillators settle with binary phases representing an optimal spin configuration for an Ising problem. . . . . 63

4.3 Phases of 20 oscillators with random $\{J_{ij}\}$ generated by `rudy -rnd_graph 20 50 10001`: (a) without SYNC; (b) with $K_s = 1$. . . . 65

4.4 Illustration of different cut sizes in a 8-vertex cubic graph with unit edge weights, and another one with random weights (rightmost). . . . . 70

4.5 Phases of oscillators solving a size-8 MAX-CUT problem without noise. . . 71

4.6 Phases of oscillators solving a size-8 MAX-CUT problem with noise. . . . 71

4.7 Simulation results from ngspice on 8 coupled oscillators. . . . . 71

4.8 A simple oscillator-based Ising machine solving size-8 cubic graph MAX-CUT problems: (a) breadboard implementation with 8 CMOS LC oscillators; (b) illustration of the connections; (c) oscilloscope measurements showing waveforms of oscillator 1$\sim$4. . . . . 72

4.9 A size-32 oscillator-based Ising machine: (a) photo of the implementation on perfboards; (b) illustration of the connectivity ; (c) a typical histogram of the energy values achieved in 200 runs on a random size-32 Ising problem; the lowest energy level is -88 and is achieved once in this case. . . . . 73

4.10 Coupled oscillators solving MAX-CUT benchmark problem G1 [2] to its best-known cut size 11624. . . . . 74

4.11 Histograms of the cut values achieved by several variants of the Ising machine, compared with the baseline results used in Tab. 4.1. . . . . 76

4.12 Speed of energy minimization for problems of different sizes. . . . . 78

4.13 Coupled oscillators coloring the states in the US map: (a) phases of oscillators evolve over time; (b) energy function (4.27) decreases during the process; (c) the resulting US map coloring scheme. . . . . 79

4.14 OIM64 Prototype. (a) photo; (b) diagram of connectivity. . . . . 80

4.15 OIM240 Prototype. (a) photo; (b) diagram of connectivity. . . . . 80

4.16 Results from OIM240: (a) energy levels of 1000 measured solutions from OIM240, on a random instance of size-240 Ising problem; (b) energy levels for 20 such random instances (showing the distances to their respective global minimum). . . . . 81

5.1 Excerpt from SPICE3's dioload.c, illustrating how every device contains code for every analysis. . . . . 84

5.2 Components of MAPP. . . . . 86

5.3 Device model prototyping flow in MAPP. . . . . 88

5.4 Tunnel diode schematic. . . . . 88

5.5 Tunnel diode I/V curve. . . . . 88

5.6 ModSpec supports multiple physical domains in the same device through the concept of the Network Interface Layer (NIL). . . . . 90

5.7 I/V curve generated from the tunnel diode ModSpec model. . . . . 91

5.8 G/V curve generated from the tunnel diode ModSpec model. . . . . 91

5.9 Circuit schematic of an oscillator made with a tunnel diode. . . . . 92

5.10 Transient simulation results from the tunnel diode oscillator in Fig. 5.9. . . 92

5.11 Screenshot from Cadence® Virtuoso®, showing Spectre transient simulation results of the circuit in Fig. 5.9. . . . . 94

5.12 Transient results from HSPICE and Xyce of the circuit in Fig. 5.9. . . . . 94

5.13 Example of problematic Verilog-A code for modelling *I–V* hysteresis. . . . 95

5.14 Contour plot of $f_2$ function in (5.12) and predicted *s*-*V* hysteresis curve based on the sign of $f_2$. . . . . 98

5.15 Results from DC sweep and transient simulation in MAPP, showing hysteresis in both $s$ and $i_1$ when sweeping the input voltage, in either type of the analyses. . . . . 99

5.16 Results from homotopy analysis in MAPP: (a) 3-D view of all the DC solutions; (b) top view of the DC solutions shows the folding in the *I–V* characteristic curve, explaining the *I–V* hysteresis from DC and transient voltage sweeps in Fig. 5.15; (c) side view of the DC solutions. . . . . 100

5.17 Forward/backward DC, transient voltage sweep responses, and homotopy analysis results from the ESD clamp model in Listing 5.3. . . . . 104

5.18 HBM test bench for an ESD clamp and transient simulation results for the voltage across the clamp. . . . . 105

5.19 Illustration of several choices of $f_2$ in RRAM model. . . . . 106

5.20 Transient results on the circuit (same as in Fig. 5.15) with a voltage source connected to an RRAM device. . . . . 109

5.21 $f_2$ function in VTEAM memristor model contains a flat region around $V = 0$ for the modelling of DC hysteresis. The proper way is to design a single solution curve of $f_2 = 0$ that folds back around $V = 0$. . . . . 110

5.22 Thermistor model with both electrical and thermal nodes. eeNIL and thermalNIL associate the I/Os in the ModSpec equations with their physical meanings. . . . . 111

5.23 Multiphysics circuit with a thermistor and its transient simulation results . 111

5.24 Code for constructing a multiphysics netlist for system in Fig. 5.23. . . . . 112

5.25 MAPP uses a master equation engine with multiple NIL interfaces to construct system-level DAEs from device models and netlist structure. . . . . 112

5.26 DC sweep on a spin valve system in MAPP. The network contains both electrical and spintronic connections (blue wires). Device diagram is adapted from [3]. . . . . 113

5.27 Diagrams of a 2-to-1 chemical reaction, a network containing three of such reactions. Transient simulations with different initial concentrations (plotted in 3-D) show that the reaction network constitutes a chemical oscillator. . . 114

5.28 Structuring of MAPP's simulation algorithms. . . . . 115

5.29 PPVs extracted by the design tools from ring oscillator latches with 2N1P and 1N1P inverters. . . . . 117

5.30 Locking range returned by the design tools on ring oscillator latches with 2N1P and 1N1P inverters. . . . . 117

5.31 Locking phase errors $|\Delta\hat{\phi}_i - \Delta\phi_i|$ within the locking range. . . . . 118

5.32 Transient simulation of the serial adder in Fig. 3.11 with oscillator latches replaced by their PPV macromodels. $\Delta\phi = 0.5$ indicates opposite phase w.r.t. REF, thus encoding phase-based 0, while $\Delta\phi = 1$ encodes 1. . . . . 119

# List of Tables

4.1 Results of oscillator-based Ising machines run on MAX-CUT benchmarks in the G-set, compared with several heuristic algorithms. Time reported in this table is for a single run. . . . . 75

4.2 Best results measured from OIM64 on size-64 rudy-generated benchmarks. Example rudy command: `rudy -toroidal_grid_2D 8 8 -random 0 1 8001 -times 2 -plus -1 > J64_01.rud`. . . . 81

5.1 NIL descriptions of some selected physical domains supported in MAPP. . 113

# Acknowledgments

First and foremost, I would like to thank my advisor and dissertation committee chair, Professor Jaijeet Roychowdhury. It goes without saying that this dissertation wouldn't exist had it not been for his support, guidance, and vision. During the seven years I have spent at Berkeley, I am continuously inspired by his scientific curiosity, his commitment to excellence, and his taste in research topics. And he has always been there for me whenever I needed help in research, career, and personal life. I wouldn't have wished for a better advisor.

I would also like to thank my other committee members, Professor Bruno Olshausen and Professor Kristofer Pister, for their help and encouragement over the years, and for being very generous to me with their time.

I feel deeply grateful and honored to have had the opportunity to collaborate with a wonderful set of researchers during my Ph.D. study. I will remember fondly the many discussions, meetings and gatherings I had with Mark Lundstrom, Dimitri Antoniadis, Luca Daniel, Supriyo Datta, Ashraf Alam, Ronald Fearing, Subhasish Mitra, Peter Bermel, Sunil Bhave, Geoffrey Coram, Colin McAndrew, Larry Nagel, Bichoy Bahr, Zheng Zhang, Kerem Camsari, Xufeng Wang, Lily Weng, Ujwal Radhakrishna, Mert Torunbalci, Juan Pablo, and Adair Gerke. And for the work on memristors, I am very thankful for the valuable discussions with and advice from Leon Chua and Stanley Williams.

I owe many thanks to the other students and post-docs in my group at Berkeley: Aadithya Karthik, Archit Gupta, Gokcen Mahmutoglu, Bichen Wu, Palak Bhushan, Shreesha Sreedhara, Leon Wu, Parth Nobel, and Jian Yao. They are not just coauthors and colleagues, but also good friends I could always turn to for support. Many thanks go to several other graduate students I used to meet almost daily in my research center and department: Ben Zhang, Wenwen Jiang, Frank Ong, Antonio Iannopollo, Marten Lohstroh, Xiangyu Yue, Baihong Jin, Yu-Yun Dai, Mehrdad Niknami, Inigo Incer, Edward Kim, Mikhail Burov, Kevin Cheang, Sen Lin, Yongjun Li, Yubei Chen, and Yumeng Liu — good friends like them have made my stay at Berkeley very enjoyable and memorable.

I am very fortunate to have the chance to learn from the several regular visitors to our group: Rajit Manohar, Alper Demir, Robert Melville, Peter Feldman, and Zubin Jacob. Discussions with them have always been fun and have helped my research greatly.

I would also like to take this opportunity to especially thank Shirley Salanio — thank her for not giving up on me after all the extra work and trouble I caused her over the years and always offering her help on the administrative work.

I feel very fortunate to have done internships with Sandia National Laboratories and Cadence Design Systems. Many thanks to Eric Keiter, Heidi Thornquist, Ting Mei, Scott

Hutchinson, Tom Russo, and Joel Phillips, for the delightful and valuable experience. I would also like to thank Colin McAndrew, for the short visit/internship opportunity at NXP Semiconductors, for pointing me to exciting research problems, and for the paper invitations.

Finally, I would like to thank my family — my parents Shaochun Wang and Limei Zhan, and my wife Lu Wang, for all the years of their love and support.

# Chapter 1

# Introduction

## 1.1 Computing beyond Moore's Law

In 1997, IBM's supercomputer Deep Blue defeated world champion Garry Kasparov in several chess games. Its mainly brute-force algorithm relied on a computing power of 11 GFLOPS (giga floating point operations per second). About two decades later, today's cell phones can generally perform more than 100 GFLOPS despite having a much smaller, more power-efficient package; mobile applications run on them are capable of beating most chess grandmasters.

Today's top supercomputers generally have peta-FLOPS of computing power, 1 million times of that of Deep Blue. How can we achieve this computing power in a hand-held device in another two decades?

Conventional wisdom says we will achieve it through semiconductor scaling. Roughly, by scaling down a transistor's dimensions (length and width) by half, its speed increases by a factor of two; all its voltages drop by half; power and area are both reduced by a factor of four. Indeed, transistor dimensions have been scaling down exponentially for decades — the number of transistors that can be integrated on a chip are doubled every 18 to 24 months, driving the rapid growth of computing power. This is Moore's law [4]. Our society has come to expect and rely on the benefits it provides.

However, Moore's law is slowing down in recent years [5]. With serious technological barriers ahead, the scaling of semiconductor transistors is coming to an end; alternative computing paradigms have to be explored. We will not provide a comprehensive survey of these approaches; instead, we provide a brief overview, organizing a few broad directions of exploration based on their time frames. This will serve as a broad background for our work.

To sustain the progress of computing in the near term, emphasis has been mainly placed on improvements to conventional CMOS transistors [6] and digital computing architectures [7]. At the individual device level, planar CMOS transistors are being replaced by multigate transistors that extend into the vertical dimension, *e.g.*, foundries today use fin field-effect transistors (FinFETs) [8], and are heavily invested in gate-all-around (GAA) FETs [9].

3D integration of such devices, and denser packaging, are also directions being actively pursued. At the system design level, parallelism in all its forms at all levels (*e.g.*, instruction-level, thread-level) has been utilized to speed up digital computation [7]. Specialization is another trend for digital hardware. From designing central processing units (CPUs) as single chips suitable for all needs, to having dedicated graphics processing units (GPUs) and digital signal processors (DSPs), specialization has been and will keep providing both speed and energy improvements. Field-programmable gate arrays (FPGAs) facilitate the implementation of custom logic operations and have also been on the rise recently. More and more application-specific integrated circuits (ASICs) are being designed and deployed for specific computational tasks, *e.g.*, machine learning [10], bio-molecular simulation [11], *etc.*, achieving more and more computing power without relying on the scaling of semiconductor transistors.

In the medium term, new device technologies that can be integrated with CMOS will come into play. As an example, optical interconnects using silicon photonics stacked with conventional CMOS [12] have been shown to hold promise in overcoming the memory bottleneck. Micro-electro-mechanical systems (MEMS) and nano-electro-mechanical systems (NEMS) can complement CMOS circuits and provide high-quality integrated components such as clocks, sensors and actuators. New memory technology such as magnetoresistive random access memory (MRAM) [13] will also enhance digital computational systems.

In the longer term, device technologies based on new materials and processes may eventually have an impact on computing. Such technologies including graphene [14], carbon nanotubes [15–17], spintronic devices [18, 19], negative capacitance FETs (NC-FETs) [20], and various memristive devices such as resistive random access memory (RRAM) [21], phase change memory (PCM) [22, 23], conductive bridge random access memory (CBRAM) [24, 25], *etc.*. The use of these novel device and material technologies has some potential for improving the speed of computation as well as reducing the power consumption. Even though several technologies have already been shown to work with CMOS, their practical large-scale deployment is still far in the future.

New computing paradigms are also essential in the longer term. Quantum computing can provide an exponential speed-up over digital computation on certain tasks [26]. Neuromorphic computing (*aka* neuro-inspired or brain-inspired computing) is another broad class of computational schemes that has attracted much research interest recently. They are analog systems designed to mimic neuro-biological architectures, and commonly rely on the novel device technologies mentioned above, *e.g.*, memristive devices [27], spintronic devices [27], *etc.*. Their exact applications are not well defined; their practical realization and deployment still await future development of emerging material and device technologies.

The oscillator-based computing schemes [28–31] we present in this dissertation are also a new computing paradigm. Our ideas on Boolean computation using oscillators (Chapter 3) change the fundamental way logic values are represented in computational systems; oscillator-based Ising machines (Chapter 4) implement a physical computation

paradigm with many advantages over digital computation for solving discrete optimization problems. But unlike the paradigms mentioned above, the realization of our schemes is not limited to the long-term future. Indeed, even with CMOS circuits, we can validate their mechanisms and principles, as well as demonstrate working prototypes. In the near term, their CMOS implementations already show potential advantages over conventional digital computational systems. Nano-device embodiments of our schemes in the future, *e.g.*, using spin-torque nano-oscillators, optical lasers and resonators, memristive-device-based nano-oscillators, bio-chemical networks, *etc.*, also offer exciting new research opportunities.

## 1.2 Oscillator-based Computation

In the recent years, there has been growing interest in using coupled oscillator networks to compute. A prevailing theme is to use oscillators to implement associative memory arrays — computational models capable of performing certain image processing tasks efficiently [32, 33]. Their oscillator-based incarnations are generally believed to be faster and more energy efficient than emulating them using digital circuits. Moreover, the recent discovery and development of various types of nano-oscillators, together with their projected scaling trends, are also motivating factors behind the growing research interest.

Associative memory is a type of (physiological or computational) memory system that enables the retrieval of data using only a partial sample of itself [32]. As such, it is naturally suitable for the pattern recognition tasks in image processing. A wide variety of proposals for oscillator-based associative memory array exist today. In almost all of them, oscillators are interconnected into a grid-like structure, coupled to their nearest neighbors; each oscillator represents a pixel in an image. Choices of oscillators include CMOS ring oscillators [34, 35], LC oscillators [36, 37], optical lasers [38], phase-locked loop (PLL) circuits [39, 40], spin-torque nano-oscillators [41–44], and oxide-based oscillators [45–49]. Depending on the oscillators of choice, the coupling between them can be resistive, capacitive, magnetic, optical, or based on active circuits using controlled sources, amplifiers, comparators, *etc.*.

There are broadly two types of such oscillator-based associative memory arrays, differing in the information encoding scheme and the system's operation. One type uses phase-based encoding [36, 39, 47]. Oscillators in these system are assumed to have the same frequency, but different coupling coefficients between them. The couplings are set based on the images or patterns to be stored. When oscillators synchronize, their relative phases represent the information retrieved.[1] The other type uses frequency-based encoding [41, 43, 45]. The coupling coefficients between oscillators are set to be the same, but each oscillator's natural frequency is set based on the stored pattern. Because different oscillators have different frequencies, they do not synchronize all together, but generate groups synchronized at different frequencies. The pattern of the group that contains the partial sample then represents the retrieved information.

[1] When the patterns to store consist of binary pixel values, the oscillator phases are commonly binarized in post-processing, using comparators or analog-to-digital converters.

Besides pattern recognition, coupled oscillators have also been shown to be suitable for some other computational tasks, such as edge detection of images [43, 50–52]. and image segmentation [53, 54]. Most related research, however, remains in the computational study stage.

## 1.3 Overview of this Dissertation

The central mechanism for all oscillator-based computing schemes is injection locking. It is a phenomenon observed in all coupled oscillator systems, in nature or man made. The study of injection locking has a long history, attracting researchers from many disciplines including biology, neuroscience, mechanical and electrical engineering. The increasing demand in communication circuits in the last several decades has greatly advanced state of the art in the modelling and simulation of oscillators and injection locking. In Chapter 2, we first go through the study of injection locking, presenting results on the phase-macromodels of oscillators, the generalized Adler's equations for analyzing injection locking, and the generalized Kuramoto model of synchronization in coupled oscillators. These analyses are the theoretical foundation of the oscillator-based computing paradigms in this dissertation.

Chapter 2 also introduces a special type of injection locking, known as sub-harmonic injection locking (SHIL), with interesting and useful features. Under SHIL, an oscillator can develop multiple stable phase-locked states. Building on this, we can encode and store phase-based logic values in oscillators, turning virtually any oscillator into a functional logic latch. In Chapter 3, we detail our ideas on using oscillators for general-purpose Boolean computation following the von Neumann architecture. But unlike the conventional implementations of digital logic circuits, our systems use oscillators as the underlying devices, and uses phase-based logic encoding. We show the feasibility of our schemes through the design, simulation and demonstration of proof-of-concept hardware prototypes, and analyze their potentials for noise immunity and energy efficiency.

The nonlinear dynamics found in coupled oscillator systems make them suitable for many computing schemes beyond the digital domain. A promising direction is to physically realize Ising machines using oscillators. The concept of Ising machines originates from the Ising model for studying ferromagnetism. The Ising model describes a network of coupled binary spins minimizing an energy function, known as the Ising Hamiltonian. As it turns out, many real-world discrete optimization problems can also be mapped to finding binary configurations that minimize the Ising Hamiltonian. Therefore, as a physical solver for this class of minimization problems, Ising machines become attractive for their high speed and energy efficiency. While existing proposal for the realization of Ising machines relies on novel device technologies (including quantum ones), we show that oscillators are very suitable for implementing the Ising spins physically. To do so, we first build upon the generalized Kuramoto model and show that a modified version of coupled oscillator system that incorporates the effects of SHIL can modify the Lyapunov function of the system such that it naturally tends to minimize the Ising Hamiltonian. We derive the principles of the

operation of oscillator-based Ising machines, study their performance by applying them to solve benchmark optimization problems, and show their feasibility and practicality by demonstrating multiple hardware prototypes built using standard CMOS technologies in Chapter 4.

The design and analyses of oscillator-based computing systems, or any novel computational paradigms, cannot be achieved without design tools for modelling and simulation. However, existing simulation platforms are not suitable for this purpose. Specifically, it is generally very difficult to add new device compact models and simulation algorithms to them. In Chapter 5, we introduce a new platform that overcomes these limitations — the Berkeley Model and Algorithm Prototyping Platform (MAPP). We explain its modular design and its use in developing new models and algorithms, providing relevant examples. In particular, we show that it is an enabling tool in the design and analysis of oscillator-based computing paradigms.

Finally, we summarize the contributions of this dissertation and discuss future work in Chapter 6.

# Chapter 2

# Injection Locking in Oscillators: Theory and Analysis

This chapter introduces phase-macromodels of oscillators and the various analyses based on them. It is the foundation of the oscillator-based computing schemes we present in the dissertation.

## 2.1 Phase Macromodels of Oscillators

A single nonlinear self-sustaining oscillator is an autonomous dynamical system, usually modelled using Differential Algebraic Equations (DAEs) [55] in the following form.

$$\frac{d}{dt}\vec{q}(\vec{x}(t)) + \vec{f}(\vec{x}(t)) = \vec{0}, \tag{2.1}$$

where $\vec{x}(t) \in \mathbf{R}^n$ represents the unknowns in the system; $\vec{q}(.)$ and $\vec{f}(.)$ are : $\mathbf{R}^n \rightarrow \mathbf{R}^n$ functions, representing the differential and algebraic parts of the DAE respectively.

A self-sustaining oscillator has a nonconstant periodic solution. That is to say, there exists a scalar time period $T_0$ and a time-varying vector $\vec{x}_s^*(t) \in \mathbf{R}^n$, such that

$$\vec{x}_s^*(t + T_0) = \vec{x}_s^*(t). \tag{2.2}$$

When this oscillator is under a time-varying perturbation modelled as $\vec{b}(t) \in \mathbf{R}^n$, its DAE can be rewritten as follows.

$$\frac{d}{dt}\vec{q}(\vec{x}(t)) + \vec{f}(\vec{x}(t)) + \vec{b}(t) = \vec{0}. \tag{2.3}$$

Suppose the solution to this DAE is $\vec{x}^*(t)$. When the perturbation is small, the oscillator's perturbed response can be approximated well [56] using the following formula.

$$\vec{x}^*(t) = \vec{x}_s^*(t + \alpha(t)), \tag{2.4}$$

where $\alpha(t)$ is a time-varying scalar representing the phase shift caused by the external input, and is governed by the following differential equation [56].

$$\frac{d}{dt}\alpha(t) = \vec{p}^T(t+\alpha(t)) \cdot \vec{b}(t), \tag{2.5}$$

where the time-varying vector $\vec{p}(t)$ is known as the Perturbation Projection Vector (PPV) [56] of the oscillator. It is $T_0$-periodic, and is a property intrinsic to the oscillator that captures its phase response to small external inputs. $\vec{p}(t)$ can be derived analytically [57] or calculated numerically [58, 59] from the oscillator's DAE without knowing any information about the input $\vec{b}(t)$. Thus, (2.2) and (2.5) are a *phase-based macromodel* of an oscillator.

## 2.2 Injection Locking and the Generalized Adler's Equation

When the external perturbation $\vec{b}(t)$ is also periodic, and it has a frequency close to the oscillator's natural frequency $\omega_0 = 2\pi/T_0$[1], injection locking can happen, and the oscillator can change its frequency to match that of the perturbation. Injection locking is the mechanism behind the synchronization phenomena in oscillators.

Injection locking can be captured using the phase-macromodel in (2.2) and (2.5) too. In this scenario, the external input is periodic or quasi-periodic. To derive the phase-macromodel in this case, we consider the input $\vec{b}(t)$ to have a period of $T^*$ (or equivalently a frequency of $\omega^* = 2\pi f^* = 2\pi/T^*$), and a phase that is either constant or slowly varying with time, represented by $\phi_{in}(t)$. This is to say, $\vec{b}(t)$ has the following form.

$$\vec{b}(t) = \vec{b}_{(2\pi)}(\omega^* \cdot t + \phi_{in}(t)), \tag{2.6}$$

where $\vec{b}_{(2\pi)}$ is $2\pi$-periodic function. In most analyses, the perturbation's phase is a constant, in which case $\vec{b}(t)$ is strictly periodic. The derivation in this chapter applies to time-varying $\phi_{in}(t)$ as well.

To derive the oscillator's response under this periodic perturbation, we first rewrite the response in (2.2) as

$$\vec{x}^*(t) = \vec{x}^*_{s(2\pi)}(\omega_0 \cdot t + \omega_0 \cdot \alpha(t)), \tag{2.7}$$

where $\vec{x}^*_{s(2\pi)}$ is a $2\pi$-periodic function with the same shape as $\vec{x}^*_s(t)$, *i.e.*, $\vec{x}^*_{s(2\pi)}(\omega \cdot t) = \vec{x}^*_s(t)$.

With the external input defined in (2.6), the differential equation for $\alpha(t)$ (2.5) can be rewritten as follows.

$$\frac{d}{dt}\alpha(t) = \vec{p}^T(t+\alpha(t)) \cdot \vec{b}_{(2\pi)}(\omega^* \cdot t + \phi_{in}(t)). \tag{2.8}$$

[1]Injection locking can also happen when the perturbation's frequency is close to integer multiples of the oscillator's natural frequency.

$\alpha(t)$ modulates the oscillator's response to the perturbation (as in (2.7)). Without loss of generality, we can simply rewrite it using another phase variable $\phi(t)$ with the following formula.

$$\alpha(t) = \frac{(\omega^* - \omega_0)\cdot t + \phi(t)}{\omega_0}, \tag{2.9}$$

or equivalently, we are defining the variable $\phi(t)$ as

$$\phi(t) = (\omega_0 - \omega^*)\cdot t + \omega_0 \cdot \alpha(t). \tag{2.10}$$

With the help of $\phi(t)$, the oscillator response in (2.7) now takes a similar form as the perturbation $\vec{b}(t)$.

$$\vec{x}^*(t) = \vec{x}^*_{s(2\pi)}(\omega^* \cdot t + \phi(t)). \tag{2.11}$$

As shown in (2.9) and (2.10), there is a one-to-one correspondence between $\alpha(t)$ and $\phi(t)$ — the transformation from to is a simple rewrite and does not rely on any assumptions. But it is useful for analyzing injection locking behaviors. For instance, if $\phi(t)$ settles to a constant, the oscillator's response $\vec{x}^*(t)$ becomes strictly periodic at $\omega^*$, matching the frequency of the external input; the value it settles to also represents the phase of the oscillator waveform under injection locking.

With the oscillator's response rewritten using $\phi(t)$, we can also rewrite the phase dynamics by deriving the differential equation governing the time evolution of $\phi(t)$.

$$\frac{d}{dt}\phi(t) = \omega_0 - \omega^* + \omega_0 \cdot \vec{p}^T_{(2\pi)}(\omega^* \cdot t + \phi(t)) \cdot \vec{b}_{(2\pi)}(\omega^* \cdot t + \phi_{in}(t)), \tag{2.12}$$

where $\vec{p}_{(2\pi)}$ is the $2\pi$-periodic version of the PPV, *i.e.*, $\vec{p}_{(2\pi)}(\omega_0 \cdot t) = \vec{p}(t)$.

To study the dynamics of (2.12), we use a concept known as the Multi-time Partial Differential Equation (MPDE) [60]. Any DAE can be formulated as a MPDE; sometimes the transformation can make it easier to solve for the DAE solutions exactly or approximately. Take the generic DAE in (2.1) as an example. Consider a related partial differential equation below.

$$\frac{\partial}{\partial t_1}\vec{q}(\hat{x}(t_1,\, t_2)) + \frac{\partial}{\partial t_2}\vec{q}(\hat{x}(t_1,\, t_2)) + \vec{f}(\hat{x}(t_1,\, t_2)) = \vec{0}, \tag{2.13}$$

where $\vec{q}(.)$ and $\vec{f}(.)$ are the same functions as in (2.1). $\hat{x}(t_1,\, t_2) \in \mathbf{R}^n$ are the unknowns in the system. (2.13) is a partial differential equation that asks us to solve for $\hat{x}(t_1,\, t_2)$ — a quantity ($n$-dimensional vector) that spans both $t_1$ and $t_2$.

Suppose we solve (2.13) and have its solution $\hat{x}^*(.,\, .)$. The theory of MPDE [60] states that the solution to the original single-time differential equation (2.1) can be written as

$$\vec{x}^*(t) = \hat{x}^*(t,\, t). \tag{2.14}$$

Now we apply the MPDE theory to the phase macromodel in (2.12). Assuming that the oscillation at $\omega^*$ happens in the fast time $t_2$ whereas the phases evolve in the slow time $t_1$, the MPDE for (2.12) can be written as

$$\frac{\partial \hat{\phi}(t_1,\, t_2)}{\partial t_1} + \frac{\partial \hat{\phi}(t_1,\, t_2)}{\partial t_2} = \omega_0 - \omega^* + \omega_0 \cdot \vec{p}^{\,T}_{(2\pi)}(\omega^* \cdot t_2 + \hat{\phi}(t_1,\, t_2)) \cdot \vec{b}_{(2\pi)}(\omega^* \cdot t_2 + \phi_{in}(t_1)). \quad (2.15)$$

Again, if we solve the PDE in (2.15) and acquire the two-dimensional PDE solution $\hat{\phi}^*(.,\, .)$, the single-dimensional solution to the original ordinary differential equation (ODE) (2.12) can be written as

$$\phi^*(t) = \hat{\phi}^*(t,\, t). \quad (2.16)$$

Because $t_2$ represents the fast time corresponding to the oscillation, the two-dimensional solution $\hat{\phi}^*(.,\, .)$ is periodic along the $t_2$ dimension [60]. We can approximate the MPDE by averaging it along the $t_2$ dimension — replacing the $t_2$-varying solution with a constant for every $t_1$. This is usually a valid approximation that does not degrade the accuracy by much, as the phase of oscillation is changing much more slowly than the oscillation itself. In this case, we approximate the two-dimensional solution $\hat{\phi}(t_1,\, t_2)$ with $\bar{\phi}(t_1)$. (2.15) becomes

$$\begin{aligned} \frac{d\bar{\phi}(t_1)}{dt_1} + 0 &= \omega_0 - \omega^* \\ &\quad + \omega_0 \cdot \int_0^{2\pi} \vec{p}^{\,T}_{(2\pi)}(\omega^* \cdot t_2 + \bar{\phi}(t_1)) \cdot \vec{b}_{(2\pi)}(\omega^* \cdot t_2 + \phi_{in}(t_1))\, dt_2 \quad (2.17) \\ &= \omega_0 - \omega^* \\ &\quad + \omega_0 \cdot \int_0^{2\pi} \vec{p}^{\,T}_{(2\pi)}(\omega^* \cdot t_2 + \bar{\phi}(t_1) - \phi_{in}(t_1)) \cdot \vec{b}_{(2\pi)}(\omega^* \cdot t_2)\, dt_2. \quad (2.18) \end{aligned}$$

To help simplify the equation, we define

$$c(t) = \int_0^{2\pi} \vec{p}^{\,T}_{(2\pi)}(t + \tau) \cdot \vec{b}_{(2\pi)}(\tau)\, d\tau. \quad (2.19)$$

$c(t)$ is a $2\pi$-periodic function. It is the cross-correlation[2] of the two functions $\vec{p}_{(2\pi)}(.)$ and $\vec{b}_{(2\pi)}(.)$.

The one-dimensional solution $\bar{\phi}(t_1)$ to (2.17) can then be used as an approximation for the solution of $\phi(t)$. Put in other words, we can rewrite the differential equation (2.12) governing the phase dynamics as

$$\frac{d}{dt}\phi(t) = \omega_0 - \omega^* + \omega_0 \cdot c(\phi(t) - \phi_{in}(t)). \quad (2.20)$$

[2]It is also known as the sliding dot product or sliding inner product.

We call equation (2.20) the Generalized Adler's Equation (GAE). Given a periodic perturbation with an input phase $\phi_{in}(t)$, the GAE governs the dynamics of $\phi(t)$, which (from (2.2)) is a quantity telling us how much the oscillator's response deviates from a strict periodic oscillation at the perturbation's frequency $\omega^*$. Put in other words, $\phi(t)$ is the phase difference from the injection-locked state.

When injection locking happens with an external periodic perturbation with a constant phase $\phi_{in}^*$, the injection-locked phase $\phi(t)$ should also become a constant. And it can be predicted by the equilibrium of (2.20). In this case, $\frac{d}{dt}\phi(t) \equiv 0$. Therefore,

$$\frac{\omega^* - \omega_0}{\omega_0} = c(\phi - \phi_{in}^*). \tag{2.21}$$

The solution of (2.21) $\phi^*$ represents the locked phase of the oscillator.

In the case of a common LC oscillator, both its waveform and PPV are sinusoidal. If it is perturbed by a voltage source through a resistive connection to an oscillating node, its PPV $\vec{p}_{(2\pi)}(.)$ is proportional to $\cos(t)$ [61]. When the perturbation is the same as the oscillator's waveform, *i.e.*, $\vec{b}_{(2\pi)}(t) = V \cdot \sin(t)$, the resulting $c(.)$ function is proportional to $\sin(.)$.

$$\frac{d}{dt}\phi(t) = \omega_0 - \omega^* + \omega_0 \cdot A \cdot \sin(\phi(t) - \phi_{in}(t)), \tag{2.22}$$

where the coupling strength $A$ is determined mainly by the $Q$ factor of the LC oscillator.

Its equilibrium equation can also be be used to predict if injection locking occurs or not with a given periodic perturbation.

$$\frac{\omega^* - \omega_0}{\omega_0} = A \cdot \sin(\phi - \phi_{in}^*). \tag{2.23}$$

Equation (2.23) is known as the Adler's equation for LC oscillators [61]. From the above analysis, it can be easily rederived from the PPV-based phase-macromodels of LC oscillators. And it is just a special case of the GAE. Beyond the analysis of LC oscillators, the GAE has many use cases; we detail them in the context of Boolean and Ising computation in Chapter 3 and Chapter 4.

One key feature of the GAE is that it can be used to analyze a special type of injection locking — sub-harmonic injection locking, or SHIL. When $\vec{b}(t)$ is a second-order perturbation, *i.e.*, it is oscillating at $2\omega^*$. We can define another $2\pi$-periodic function $\vec{b}_{2\pi}^{(2)}(t)$, such that $\vec{b}_{2\pi}(t) = \vec{b}_{2\pi}^{(2)}(2t)$. If we further assume that the PPV is also oscillating at twice its natural frequency, *i.e.*, $\vec{p}_{2\pi}(t) = \vec{p}_{2\pi}^{(2)}(2t)$, it can be proven that $c(t)$ is also a second-order

oscillation.

$$c(t) = \int_0^{2\pi} \vec{p}^{(2)T}_{(2\pi)}(2t+2\tau)\cdot\vec{b}^{(2)}_{(2\pi)}(2\tau)\,d\tau \tag{2.24}$$

$$= \frac{1}{2}\int_0^{4\pi} \vec{p}^{(2)T}_{(2\pi)}(2t+\tau)\cdot\vec{b}^{(2)}_{(2\pi)}(\tau)\,d\tau \tag{2.25}$$

$$= \int_0^{2\pi} \vec{p}^{(2)}_{(2\pi)T}(2t+\tau)\cdot\vec{b}^{(2)}_{(2\pi)}(\tau)\,d\tau \tag{2.26}$$

$$\triangleq c^{(2)}(2t), \tag{2.27}$$

where $c^{(2)}(.)$ is a $2\pi$-periodic function, making $c(.)$ is $\pi$-periodic.

In this special case, the phase dynamics (2.20) can then be written as

$$\frac{d}{dt}\phi(t) = \omega_0 - \omega^* + \omega_0\cdot c^{(2)}(2(\phi(t)-\phi_{in}(t))). \tag{2.28}$$

with its equilibrium equation written as

$$\frac{\omega^*-\omega_0}{\omega_0} = c^{(2)}(2(\phi-\phi^*_{in})). \tag{2.29}$$

If we again assume $c^{(2)}(.)$ is sinusoidal, now between 0 and $2\pi$, there can be two solutions of $\phi$, representing the bistable phase-locked states under second-order SHIL. As $c^{(2)}(.)$ is $2\pi$-periodic, it can be easily proven that if one solution exists, the other one separated by $\pi$ is also a solution. Suppose one solution is $\phi^*_1$, satisfying

$$\frac{\omega^*-\omega_0}{\omega_0} = c^{(2)}(2(\phi^*_1-\phi^*_{in})). \tag{2.30}$$

Then for $\phi^*_2 = \phi^*_1+\pi$,

$$\frac{\omega^*-\omega_0}{\omega_0} = c^{(2)}(2(\phi^*_1-\phi^*_{in})) \tag{2.31}$$

$$= c^{(2)}(2(\phi^*_1-\phi^*_{in})+2\pi) \tag{2.32}$$

$$= c^{(2)}(2((\phi^*_1+\pi)-\phi^*_{in})) \tag{2.33}$$

$$= c^{(2)}(2(\phi^*_2-\phi^*_{in})). \tag{2.34}$$

Therefore, $\phi^*_2$ is also a solution. The two solutions are separated by $\pi$, or $180^\circ$.

## 2.3 Models of Coupled Oscillators

The Kuramoto model is commonly used to study the synchronization of coupled oscillators [62].

$$\frac{d}{dt}\phi_i(t) = \omega_i + \frac{K}{n}\sum_{j=1,\ j\neq i}^{n}\sin(\phi_j(t)-\phi_i(t)), \tag{2.35}$$

where $\phi_i$ is each oscillator's phase; $\omega_i$ is each oscillator's intrinsic natural frequency; $n$ is the number oscillators; $K$ represents the strength of coupling.

It was first proposed by Yoshiki Kuramoto [63]. It was based on inspiration from biological and chemical oscillator systems [62], and Kuramoto was surprised [64] that the model turned out to be able to capture the behavior of many other oscillator systems, such as synchronized neurons [65] and coupled arrays of Josephson junctions [66]. The model describes an all-to-all connectivity between oscillators. Later variants can support arbitrary connectivity and coupling coefficients [67].

Many oscillator-based computing schemes and neuroscience studies start with the Kuramoto model and its variants. As they are all high-level mathematical models of the synchronization phenomenon in general, they were not derived from or connected with lower-level oscillator models. Our PPV-based phase-macromodels (Sec. 2.1) and the GAE model (Sec. 2.2) can immediately and naturally fill this gap. Based on them, we show how this mathematical model of synchronization is connected to the differential equations of individual oscillators, particularly how the phase interference function (*i.e.* the $\sin(.)$in the Kuramoto model) is derived from oscillator equations. The result is a more general version of the Kuramoto model derived from the ground up.

In a coupled oscillator system, the perturbation to an oscillator comes from its connections to the other oscillators. We can write the phase dynamics of the $i$th oscillator as

$$\frac{d}{dt}\phi_i(t) = \omega_i - \omega^* + \omega_i \cdot \sum_{j=1,\ j\neq i}^{n} \left[\vec{p}_{ij}^{\,T}(\omega^* \cdot t + \phi_i(t)) \cdot \vec{b}_j(\omega^* \cdot t + \phi_j(t))\right]. \tag{2.36}$$

In equation (2.36), $\omega_i$ is the frequency of this oscillator, $\omega^*$ is the central frequency for the coupled oscillators when they synchronize. $\vec{p}_{ij}(.)$ represent the entries (or sub-vector) of the $2\pi$-periodic PPV of the $i$th oscillator, with perturbation from the $j$th oscillator, $\vec{b}_j(.)$ is the $2\pi$-periodic perturbation from the $j$th oscillator.

We can apply the same Adlerization technique as in Sec. 2.2, and define

$$c_{ij}(t) = \int_0^{2\pi} \vec{p}_{ij}^{\,T}(t+\tau) \cdot \vec{b}_j(\tau)\, d\tau. \tag{2.37}$$

Then from the MPDE theory, (2.36) can be approximated well by

$$\frac{d}{dt}\phi_i(t) = \omega_i - \omega^* + \omega_i \cdot \sum_{j=1,\ j\neq i}^{n} c_{ij}(\phi_i(t) - \phi_j(t)). \tag{2.38}$$

We call (2.38) the Generalized Kuramoto model. The original Kuramoto is just a special case with $c_{ij}(.) = -\sin(.)$. The Generalized Kuramoto model can be applied to any type of nonlinear oscillators — we can start with the oscillator's DAE, extract its PPV-based

phase-macromodel, combine it with the type of coupling to derive the GAE model for an individual oscillator or the Generalized Kuramoto model for coupled oscillator systems. We use this procedure repeatedly in our study in Chapter 3 and Chapter 4, and discuss the numerical implementation of this analysis flow and the corresponding design tools in Chapter 5.

Furthermore, under certain circumstances explained below, the phase dynamics described by the Generalized Kuramoto model are governed by a global Lyapunov function.

$$E(\vec{\phi}(t)) = -\sum_{i\neq j} C_{ij}(\phi_i(t)-\phi_j(t)) - 2\sum_{i=1}^{n}\frac{\omega_i-\omega^*}{\omega_i}\phi_i(t), \tag{2.39}$$

where $C_{ij}(t)$ is defined as follows.

$$C_{ij}(t) = \int_0^t c_{ij}(\tau)\, d\tau + C_{0ij}, \tag{2.40}$$

with $C_{0ij}$ being an arbitrary constant.

A Lyapunov function is a scalar function used to study the stability of an equilibrium of a dynamical system. A global Lyapunov function applies to all the system's equilibria [68]; it is a quantity the system always tends to minimize.

To show that (2.39) is a global Lyapunov function of the generalized Kuramoto model, we differentiate $E$ with respect to $\vec{\phi}$. We observe that $E$ is the sum of $(n^2-n)$ number of $C_{ij}()$ terms. Among them, for any given index $k$, variable $\phi_k$ appears a total of $2\cdot(n-1)$ times. It appears $(n-1)$ times as the subtrahend inside $C_{ij}()$, and the other $(n-1)$ times as the minuend inside $C_{ij}()$. So when we differentiate $E$ with respect to $\phi_k$, we have

$$\begin{aligned}
\frac{\partial E(\vec{\phi}(t))}{\partial \phi_k(t)} =& -\sum_{l=1,\ l\neq k}^{n}\frac{\partial}{\partial\phi_k(t)}\left[C_{ij}(\phi_k(t)-\phi_l(t))\right] \\
&-\sum_{l=1,\ l\neq k}^{n}\frac{\partial}{\partial\phi_k(t)}\left[C_{ij}(\phi_l(t)-\phi_k(t))\right] \\
&-\frac{\partial}{\partial\phi_k(t)}\left[2\frac{\omega_k-\omega^*}{\omega_k}\phi_k(t)\right] && (2.41)\\
=& -\left[\sum_{l=1,\ l\neq k}^{n} c_{ij}(\phi_k(t)-\phi_l(t)) - \sum_{l=1,\ l\neq k}^{n} c_{ij}(\phi_l(t)-\phi_k(t))\right] \\
&-2\frac{\omega_k-\omega^*}{\omega_k} && (2.42)\\
=& -2\sum_{l=1,\ l\neq k}^{n} c_{ij}(\phi_k(t)-\phi_l(t)) - 2\frac{\omega_k-\omega^*}{\omega_k} && (2.43)\\
=& -\frac{2}{\omega_k}\cdot\frac{d\phi_k(t)}{dt}. && (2.44)
\end{aligned}$$

The derivation assumes $c_{ij}(.)$ is an odd function symmetric with respect to the origin, *i.e.*, $c_{ij}(-x) = -c_{ij}(x)$. Base on the derivation,

$$\frac{\partial E(\vec{\phi}(t))}{\partial t} = \sum_{k=1}^{n} \left[ \frac{\partial E(\vec{\phi}(t))}{\partial \phi_k(t)} \cdot \frac{d\phi_k(t)}{dt} \right] \tag{2.45}$$

$$= -\frac{2}{\omega_k} \cdot \sum_{k=1}^{n} \left( \frac{d\phi_k(t)}{dt} \right)^2 \leq 0. \tag{2.46}$$

$E(\vec{\phi}(t))$ is non-increasing over time; (2.39) is indeed a global Lyapunov function of the system.

# Chapter 3

# Boolean Computation using Oscillators

In this chapter, we present our approach of implementing general-purpose Boolean computational systems using oscillators with phase-based logic encoding.

Phase-based or temporal-based logic encoding, although rarely discussed in the context of computation, is not a novel idea by itself. It is widely used in communication systems and is commonly preferred over level-based logic encoding, *e.g.*, phase modulation (PM) in radio is often considered advantageous compared with amplitude modulation (AM) for transmitting signals through noisy channels. Using it in computation is a natural direction to explore.

Indeed, the notion of computation with phase logic is almost as old as digital computation itself. In the 1950s, Eiichi Goto and John von Neumann each filed patents describing their ideas of using resonant circuits to realize phase-encoded computation. Compared with the vacuum tube technology then used in computing, their circuits were cheap and reliable, and were thus very attractive. Goto's phase-encoded computers were momentarily popular and commercially successful in Japan. But these implementations were quickly overshadowed by the rise of transistors and the level-based logic computing systems built with them — the duo have been dominating computation ever since. Sec. 3.1 provides a brief overview of this history of phase-encoded logic computation, explaining Goto's and von Neumann's proposals, and examining the historical circumstances for their rise and fall.

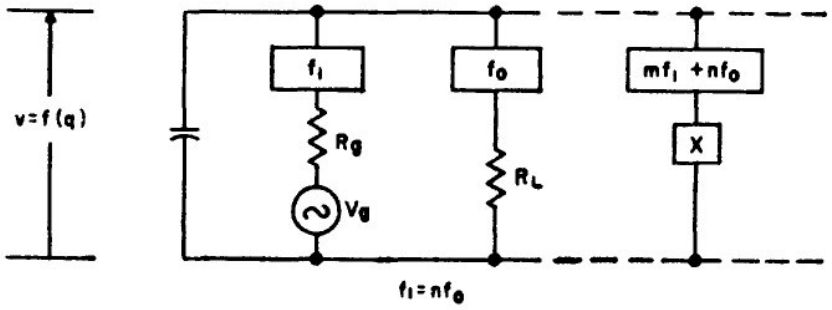
Then we take a fresh look at their ideas and present improvements that can overcome the major limitations of their previous implementations. Instead of using their devices based on passive filters, we show that DC-powered self-sustaining nonlinear oscillators can be the substrate for phase-encoded Boolean computation. Specifically, we show how the analyses in Chapter 2 enable the use of any oscillator — including electronic, spintronic, biological, optical and mechanical ones — to serve as a logic latch. Pairing these latches with phase-based logic gates, we can realize oscillator-based finite state machines, which are the core of general-purpose digital computers.

We detail our approach in Sec. 3.2, design and demonstrate such computing systems using various oscillators in Sec. 3.3, then discuss the noise immunity and potential energy

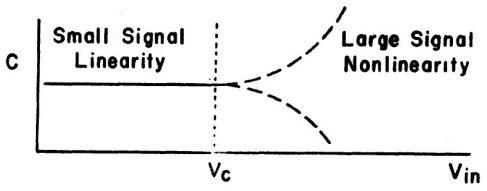
efficiency advantages of our scheme in Sec. 3.4.

## 3.1 History of Phase-encoded Computation

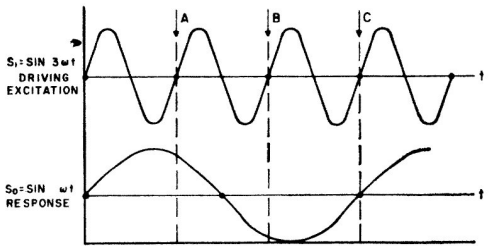
In 1954, John von Neumann and Eiichi Goto each filed a patent describing their ideas on phase-encoded computation. Here in this section, we provide a brief sketch of of them, with an emphasis on von Neumann's proposal.




(a) AC-powered subharmonic generator features multiple phase-shifted stable states [69, Fig. 2].




(b) Multistability beyond a critical pump amplitude, $V_c$ [69, Fig. 4].

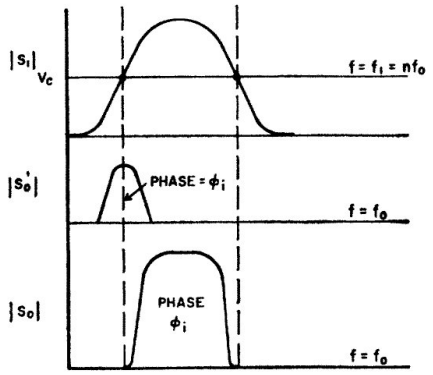



(c) Example: discriminating between subharmonic steady states [69, Fig. 6].

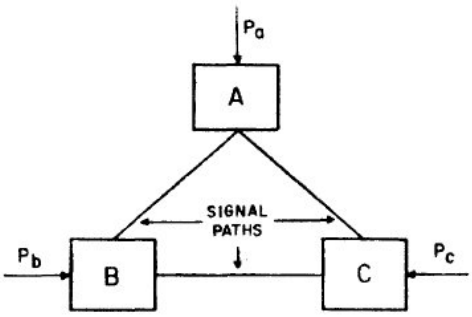
**Fig. 3.1:** von Neumann's basic phase-based latch: a nonlinear AC-pumped circuit with multiple subharmonic steady states.

In his patent [69–71], von Neumann started with a key observation: the circuit in Fig. 3.1(a) can feature *two (or more) distinct oscillating steady states*, which can be used to *store two (or more) logical states* stably. The circuit features a *lossless, nonlinear, charge-controlled capacitor* (shown towards the left of Fig. 3.1(a)), together with bandpass filters used to isolate an AC power source or *pump* ($V_g$), and an output load ($R_L$), from the several frequencies simultaneously present in the capacitor's terminal voltage. When the amplitude of the AC pump voltage waveform, assumed sinusoidal at frequency $f_1$, is larger than a critical threshold $V_c$ (Fig. 3.1(b)), the voltage waveform across the capacitor can feature components that are *integer sub-multiples* of $f_1$, *i.e.*, a sine wave at $f_0 = f_1/n$ which is a sub-harmonic of the $f_1$ AC pump. Furthermore, as depicted in Fig. 3.1(c), a generated sub-

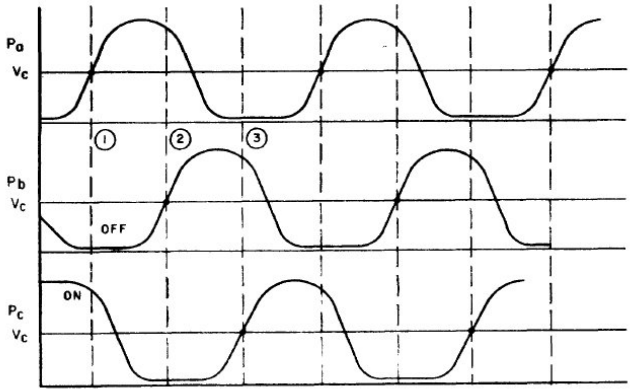
harmonic can be in *one of several distinct phase relationships* with respect to the waveform of the AC pump. Each distinct phase relationship can be used to encode a logic state. The circuit functions, effectively, as a *latch* that can store a logic value. The sub-harmonic and multi-stability properties of the circuit in Fig. 3.1(a) can be inferred from an elegant formula, the *Manley-Rowe relationships* [69, 70, 72].




(a) Amplitude modulation of the AC pump to capture an input logic value [69, Fig. 7].




(b) A ring of latches can retain a phase-logic state permanently [69, Fig. 11].




(c) Phase-shifted amplitude modulation waveforms driving the ring of latches [69, Fig. 11].

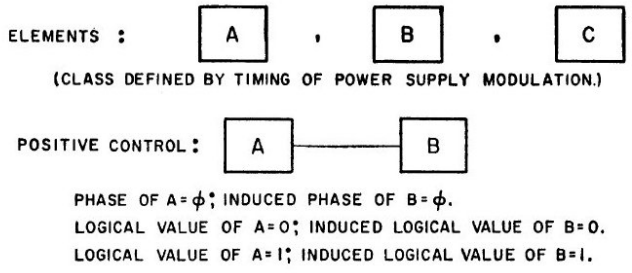
**Fig. 3.2:** von Neumann's scheme for setting a latch to an input state and retaining it.

Building on this principle, von Neumann's explained the key mechanisms for his phase-encoded logic latch and computational system.

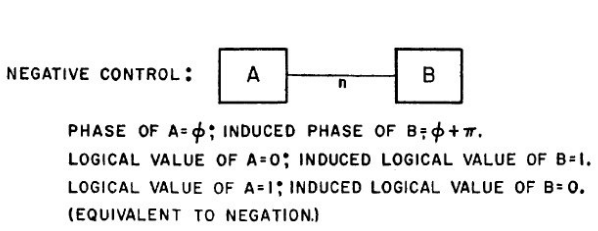
*Setting a phase-encoded latch to an input logic state*: Having devised a latch circuit capable of storing logical values encoded in phase, von Neumann considered the question of setting a latch to a desired logic state supplied as input. He proposed a scheme based on modulating the amplitude of the AC pump slowly with a waveform similar to the uppermost

graph of Fig. 3.2(a). When this modulation waveform is low, the latch is, effectively, turned off; as the modulation increases and magnitude of the AC pump crosses the critical threshold $V_c$, the latch turns on and settles to one of the possible logic states. von Neumann suggested that if a desired logic value (encoded in phase) were to be introduced as an input to the latch just as it was turning on, the latch would settle to (the sub-harmonic phase corresponding to) the same logic value. This is depicted in the middle and bottom graphs of Fig. 3.2(a).

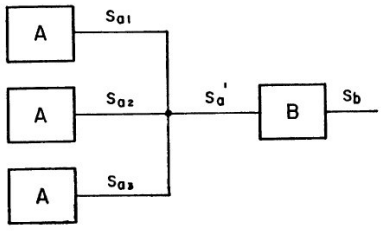
*Holding on to the logic state*: A problem with the above input-latching scheme is, of course, that the latch is turned off periodically, thereby losing its stored state. von Neumann's solution was a ring of latches (Fig. 3.2(b)), with each latch's AC pump modulated by a phase shifted version of its predecessor's AC modulation (Fig. 3.2(c)). The ring operates in merry-go-round fashion, with each succeeding latch turning on, capturing its predecessor's logic state and retaining it as the predecessor subsequently turns off. At any given time, one latch is always on, hence the logic state is never lost.




(a) Pump modulation timing defines latch-ring classes [69, Fig. 13].




(b) NOT: through connection between opposite latch-ring classes [69, Fig. 13].




(c) MAJORITY: add phase-encoded waveforms [69, Fig. 14].

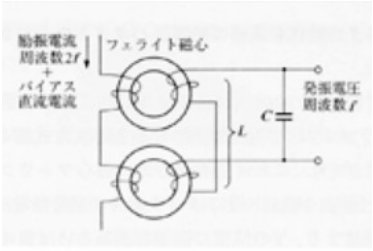
**Fig. 3.3:** von Neumann's scheme for combinatorial logic in phase.

*Combinatorial operations for phase-encoded logic*: Next, von Neumann turned to the problem of realizing arbitrary Boolean operations using phase-encoded logic. Noting that two operations, NOT and MAJORITY, constitute a logically complete set [1], he provided especially elegant means of realizing them [69]. NOT is obtained simply by a through connection between latches with different pump modulation phases while MAJORITY is obtained simply by adding the waveforms of the three inputs together (Fig. 3.3(c)).

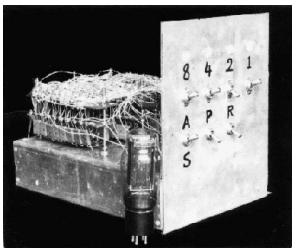
[1] *i.e.*, any Boolean function can be realized using compositions of functions in this set.

Having devised phase-based realizations of the latch-ring and the logically complete combinatorial function set NOT and MAJORITY, together with a consistent timing scheme provided by the phase shifted pump modulation waveforms, von Neumann had developed all the basic components needed to make state machines and general computing architectures.

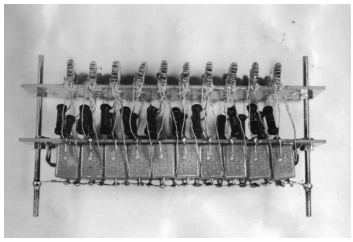
Around the same time von Neumann patented his idea on phase-encoded computation, Eiichi Goto, then a graduate student in the laboratory of Takahasi Hidetosi at the University of Tokyo, filed a patent in Japan [73] describing a similar invention. It is unclear to us who came up with the idea first; there is no known connection between the two patents either. Goto's resonant circuit is known as the parametron. As shown in Fig. 3.4(a), a parametron consists of an inductor-capacitor tank; the inductance is modulated by an external AC pump at twice the frequency of the tank's resonant frequency. The AC pump can excite a parametric oscillation state with a bistable phase in which a binary logic value can be encoded and stored. Parametrons can also be used to implement logic gates with phase-based negation and majority functionalities, and can be interconnected to perform any Boolean logic operation. Fig. 3.4(b) shows a prototype of an adder made of parametrons (the parametron module is shown in Fig. 3.4(c)). In 1958, a parametron-based computer named PC-1 was prototyped in the University of Tokyo. Later versions of parametron-based computers were in fact popular in Japan in the 1950s and 1960s due to their simplicity and reliability [73–77].




(a) Circuit schematic of Parametron, from [78].




(b) Photo of the first parametron adder, from [79].



(c) Parametrons used in the adder, from [79].

**Fig. 3.4:** Goto's design of parametrons and parametron-based computing systems.

However, the advent of transistors and integrated circuits quickly led to the demise of Goto and von Neumann's computers, as the devices and circuits their proposals relied on could not compete with the level-based logic computational systems using transistors. A key reason for their lack of competitiveness was *size and miniaturizability*. Specifically, the filters in both Fig. 3.1(a) and Fig. 3.4(a) require inductors, which are large and bulky compared with semiconductor transistors, particularly today's nanoscale MOS devices.[2] Capacitances much larger than pF also take far more chip area than transistors do. Another

[2]Indeed, only over the last two decades or so have inductors been integrated on chip at all, and even so, they consume much area, suffer from low $Q$ (high loss), and contribute disproportionately to chip design and fabrication cost.

related reason was *lower operating speed*, stemming not only from larger component sizes, but also from inherent features of von Neumann's scheme (*e.g.*, periodic turn-on transients and delays in NOT that made phase-based logic slower than transistorized level-based logic. Later attempts (from the 1980s to the present) used superconducting Josephson junction devices [80], which are fast, but still limited in terms of miniaturizability and practical deployment at room temperature.

Because physical implementations have not, to date, been miniaturizable or large-scale integrable, issues of noise immunity and energy efficiency for phase-based logic did not come to the fore. Yet, as we show in Sec. 3.4, these constitute significant competitive advantages of phase-based logic [3]. With the extensions described in Sec. 3.2, the limitations of large size and lack of integrability in Goto and von Neumann's circuits (Fig. 3.1(a) and Fig. 3.4(a)) are removed, making phase logic a serious and promising alternative computing scheme.

## 3.2 PHLOGON: PHase-based LOGic using Oscillatory Nano-systems

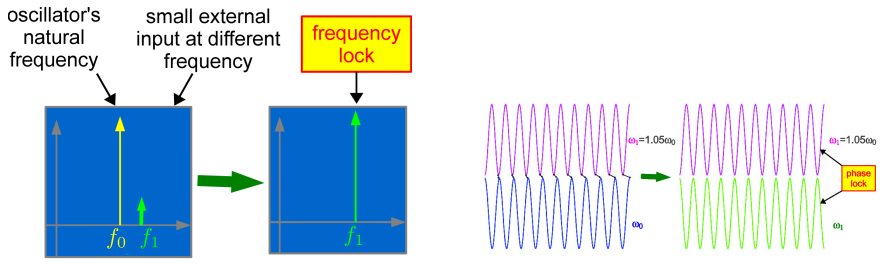
In Chapter 2 we have introduced sub-harmonic injection locking (SHIL) and the multiple stable phase-locked states it induces in oscillators. These states can be used to store phase-based logic values, essentially making any self-sustaining nonlinear oscillator suitable as a logic latch. We explain the operation of oscillator latches in Sec. 3.2.1, then discuss the implementation of phase-based logic gates and the system-level design of phase-encoded Boolean systems in Sec. 3.2.2. Then in Sec. 3.2.3, we discuss our approach's differences from the other oscillator-based computing schemes mentioned in Sec. 1.2, and its differences from Goto/von Neumann's original designs. Our schemes of oscillator-based Boolean computation are collectively termed PHLOGON — PHase-based LOGic using Oscillatory Nano-systems.

### 3.2.1 Oscillator Latches with Phase Logic

The suitability of oscillators for use as phase-based logic latches depends critically on their ability to 1) feature multiple stable states, and 2) be set to, and hold on to, a desired logical state. Both can be achieved through injection locking (IL) and sub-harmonic injection locking (SHIL).

While Chapter 2 introduces IL and SHIL mathematically, here we recapitulate their concepts in a more intuitive way. When an oscillator with a natural frequency of $f_0 = \omega_0/(2\pi)$ is perturbed by an external input at $f_1 \approx f_0$, through IL, the oscillator's response can lock on to that of the input in both frequency (shown in Fig. 3.5(a)) and phase (show in Fig. 3.5(b)). The analysis in Sec. 2.2 also states that, when the input frequency changes to $f_1 \approx 2f_0$, SHIL can happen. Under SHIL, an oscillator locks to a sub-harmonic of the input,

[3] Indeed, the reliability of phase-based logic was noted early in Japan

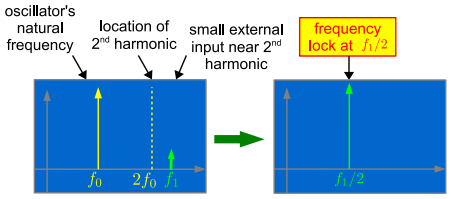



(a) Frequency lock under injection locking.

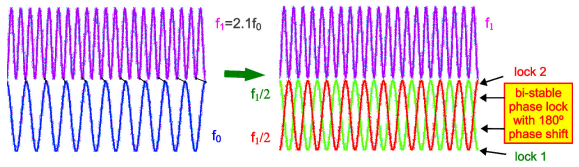
(b) IL also features phase lock.

**Fig. 3.5:** Illustration of frequency and phase lock of injection locking.

and can develop bistable phase-locked states.[4] In other words, our analysis in Chapter 2 shows that the oscillator response will be exactly half the frequency of the input if SHIL occurs, and the two phase-locked responses will be separated by 180°, as illustrated in Fig. 3.6(a) and Fig. 3.6(b). The two phase-locked oscillation states are suitable for encoding and storing a phase-based binary logic value. And in our scheme, we call the second-order input signal (at $f_1 \approx 2f_0$) responsible for inducing this bistability a synchronization signal, or SYNC.




(a) Under SHIL, the oscillator's frequency locks to the sub-harmonic of the input.




(b) Under SHIL, phase lock features multistability. When $f_1 \approx 2f_0$, the two stable states have a 180° phase difference.

**Fig. 3.6:** Illustration of frequency and phase lock of sub-harmonic injection locking.

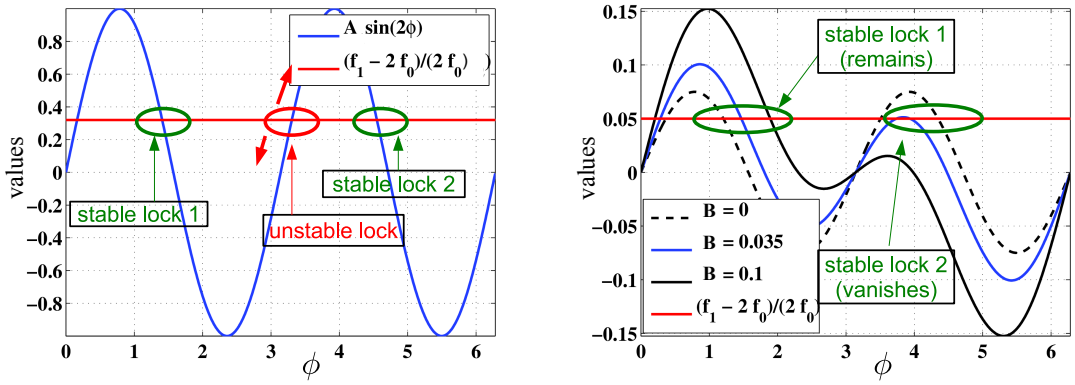
An oscillator can be influenced by both first-order IL (at the fundamental frequency) and SHIL simultaneously. Chapter 2 has shown how to use the Generalized Adler Equation (GAE) to gain insights into IL and SHIL properties. In this case, the GAE equilibrium equation can be written as follows.

$$\frac{f_1 - 2f_0}{2f_0} = A \cdot \sin(2\phi) + B \cdot \sin(\phi - \phi_{in}). \tag{3.1}$$

Under this combination of IL and SHIL, the oscillator is frequency locked at a fundamental frequency of $\frac{f_1}{2}$. In (3.1), $A$ and $B$ describe the strength of phase perturbation at this

[4] We have shown that when SYNC is at $f_1 \simeq mf_0$, the phase of the oscillator may feature $m$ distinct stable states and the analysis performed here is still applicable[81, 82]. We will henceforth take $m = 2$ (binary encoding) to illustrate all the main ideas.

fundamental frequency and its second harmonic $f_1$; they can be calculated based on the PPV of the oscillator and the magnitude of the perturbations (Chapter 2). Solutions of $\phi$ of (3.1), if they exist, are the possible phase shifts of the oscillator latch's waveforms under both IL and SHIL. Considerable insight into the number, nature and behaviour of these solutions can be obtained graphically, by plotting the left- and right-hand-sides of the GAE equilibrium equation separately and looking for intersection points.




(a) (3.1) has 2 stable solutions in the absence of a (logic) fundamental frequency input.

(b) Acquisition of input phase: phase bistability of (3.1) vanishes.

**Fig. 3.7:** GAE equilibrium equation (3.1) establishes multistability and input phase acquisition properties of oscillator latches.

Fig. 3.7(a) plots the left- and right-hand-sides (flat red and sinusoidal blue traces, respectively) of (3.1) when $B = 0$, *i.e.*, no (logic) fundamental frequency input is present for the oscillator. There are 4 intersections between the two traces, corresponding to 4 solutions of (3.1). Of these, the first and third intersections (from the left) can be shown to be dynamically unstable[5]; but the second and fourth intersections correspond to *two distinct stable oscillations, sub-harmonically locked with phases separated by* $180^\circ$. The oscillator features bistability for storing a binary logic bit.

Fig. 3.7(b) plots the same left-hand-side (flat red trace), but overlays several traces for the right-hand-side of (3.1), corresponding to $B$ values 0, 0.035 and 0.1, with the latter two values representing two different strengths of an incoming signal at the fundamental frequency $\frac{f_1}{2}$ with $\phi_{in} = 0$ — a signal representing a reference phase-based logic value 0 or 1. As can be seen, the first stable intersection remains relatively unaffected as the strength of the incoming input changes, but the second intersection vanishes, structurally, for $B$ value 0.1. This implies that the oscillator acquires the input's logic state. After acquisition, as $B$ is reduced to zero, the GAE can be used to analyze the dynamics of $\phi$ and show that the acquired logic state is held, even though the second stable intersection is restored.

[5] *i.e.*, they are physically unrealizable in the presence of perturbations, noise or variability.

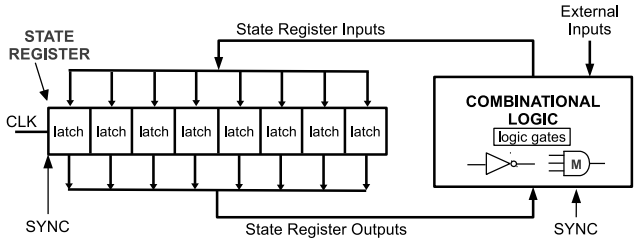



**Fig. 3.8:** A general FSM. SYNC is used to develop multi-stability for encoding phase logic; NOT/MAJORITY gates are for combinational logic operations.

From above, the SHIL phenomenon enables the oscillator to develop multiple, well-defined, stable states that can be used to encode and store logic. It also shows how the oscillator latch can acquire phase and switch between logic states according to external logic inputs.

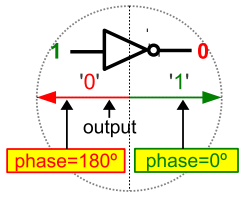
Again, it is worth mentioning that the mechanisms and analyses above are not specific to any particular type of oscillators. As we show through PHLOGON examples using various oscillator technologies in Sec. 3.3, the mechanism is general and allows virtually all nonlinear oscillators to be used as practical logic latches. Moreover, an important property of injection-locked oscillators is that they feature excellent variability and noise resistance properties. Our use of self-sustaining oscillators and injection locking to realize phase logic can take advantage of this property to achieve low-level noise, interference and variability resistance; such potentials are discussed in Sec. 3.4.

### 3.2.2 General-purpose Computing with Phase Logic

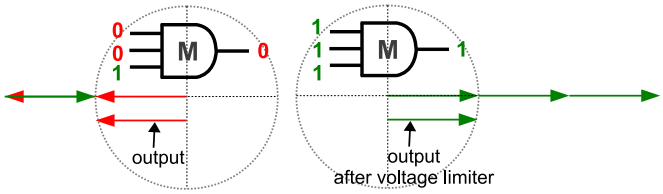
The central unit of a general-purpose computer is a finite-state machine (FSM) [71]. As is shown in Fig. 3.8, latches and combinational logic blocks are the key components of an FSM. Here, we first describe how to build combinational logic blocks using phase logic.

We realize combinational logic operations in a manner almost identical to Goto/von Neumann's technique, using NOT and MAJORITY operations. Phase logic enables elegant implementation of these two operations: NOT is simply inversion and MAJORITY can be implemented by addition. These can be explained using phasors, as is demonstrated in Fig. 3.9. Note that the phasor explanation only demonstrates the validity of using inversion and addition for NOT and MAJORITY operations, the actual implementation of the logic gates is flexible and dependent on the type of oscillator that is used. The possibility of utilizing nano-scale devices that can naturally perform phase NOT/MAJORITY operation needs exploration for different physical domains. Also, to take these implementation ideas one step further, we can use oscillator latches and SYNC signal at the output stage of logic gates to prevent variability at the inputs from accumulating and propagating. This approach also makes it possible to construct robust large-scale combinational logic blocks.

As for the latches and registers in FSMs, we have already discussed the principles of




(a) Inverting input signal performs NOT operation in phase logic.

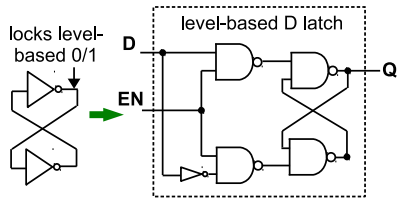



(b) A three-input MAJORITY gate uses addition to perform MAJORITY operation of input signals in phase logic.

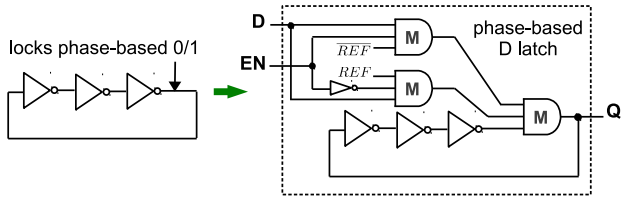
**Fig. 3.9:** Phase-domain plots illustrating NOT and MAJORITY operations.

their operation in Sec. 3.2.1, showing how a phase-based logic bit can be stored through the mechanism of SHIL and how the bit can be set or flipped through first-order (fundamental frequency) IL. But to use them inside practical computing systems, we need latches with well-defined input/output logic behaviors, such as D latches or SR latches. Building on the principles in Sec. 3.2.1, a phase-based D latch can be implemented in multiple ways.

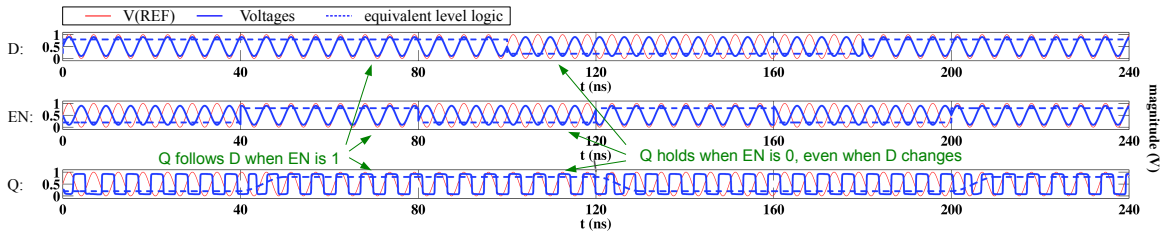
The simplest way is to gate the input signal (D) with a switch (*e.g.* a transmission gate). In this case, when the switch is off and input is isolated, the latch holds on to its stored logic value; when it is on, the latch acquires the logic value of the input. The switch can be controlled by an enable signal (EN), which can simply be a high/low voltage. In state machines, EN is normally the same as clock (CLK); it can be the only voltage-level-based signal in the system whereas others can all be encoded in phase.




(a) Level-based D latch derives from bistable level-latching device.




(b) Phase-based D latch derives from bistable phase-based bit storage device (*e.g.* CMOS ring oscillator).




(c) Simulation results of phase-based D latch implemented as a CMOS ring oscillator.

**Fig. 3.10:** Design and implementation of phase-based D latch.

Another approach can encode EN and CLK using phase-based logic as well. With the help of phase-based logic gates, it is not hard to implement logic operations that are equivalent to

the EN switch discussed above. Fig. 3.10(b) shows a diagram of a D latch based on a CMOS ring oscillator, implemented entirely using phase-based logic; the same design making use of majority gates is applicable to other types of oscillators as well. The design is inspired by the way extra logic gates are used to turn a back-to-back inverter system that can store a bit into a level-based D latch that can be controlled by D and EN signals (Fig. 3.10(a)). It is the equivalent design using phase-based logic encoding. Fig. 3.10(c) shows waveforms from SPICE-level simulation of this D latch. By analyzing the waveforms of Q, D, EN with REF, we see that the system achieves the functionality of a transparent D latch in phase logic. Tying two such transparent D latches together results in an edge-triggered master-slave D flip-flop — a key component in a FSM.

With oscillator latches and flip-flops to store phase-based bits, together with logic gates to perform combinational operations, we now have all the components to build general-purpose computing systems. Without loss of generality, here we use CMOS ring oscillators as an example to show how such computing systems can be built.

Fig. 3.11 shows a one-bit FSM (with its state transition graph shown in Fig. 3.12) — a serial adder made of the D flip-flop and a MAJORITY/NOT-based full adder. Simulation results are shown in Fig. 3.13, where we add $a = 101$ with $b = 101$ sequentially during three clock cycles. From Fig. 3.13 we can see *cin* is held stable every time CLK level is low (translates to having opposite phase as REF). During this time $cout = 101$ and $sum = 010$ can be read out sequentially. In the full system design their values can then be latched using other registers and connected to following stages in the system, or transformed to level-based logic if the system needs to be connected to conventional computation systems or display blocks.

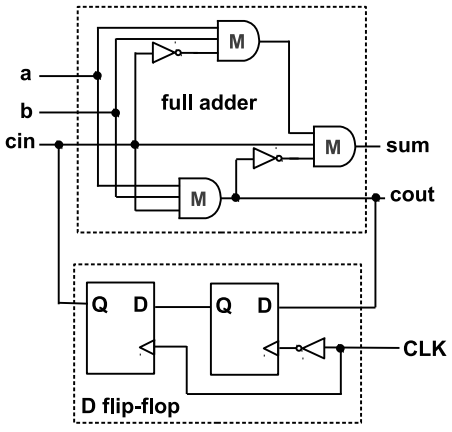



**Fig. 3.11:** Serial adder.

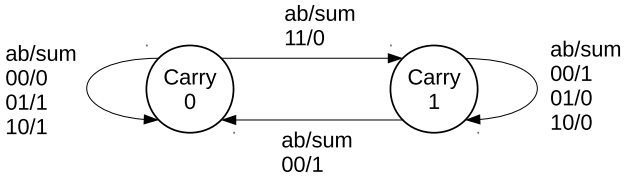



**Fig. 3.12:** State transition diagram of a serial adder.

Even though our scheme differs from conventional digital computers in the fundamental way logic is encoded, the system-level design concepts and procedures are similar. As a result, all the logic synthesis and timing analysis theories and tools can potentially still be used with only minor modifications, immediately enabling complex, large-scale system

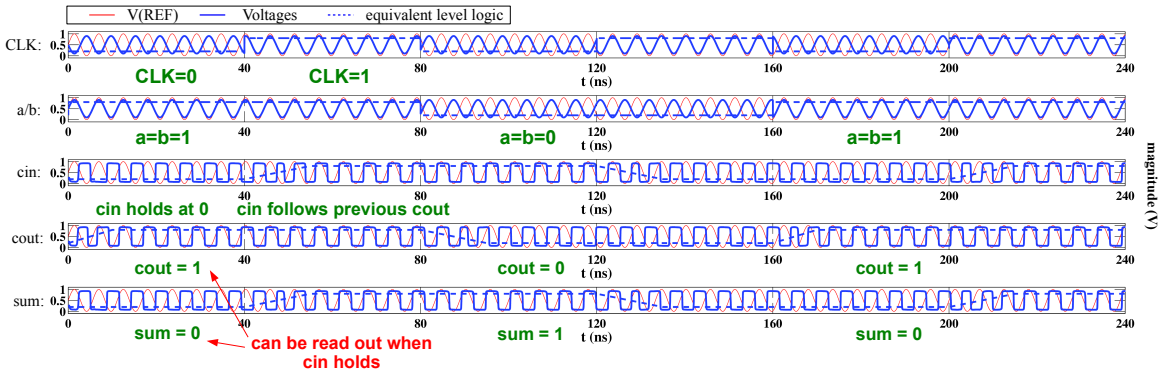



**Fig. 3.13:** Waveforms from adding $a = b = 101$ with serial adder in Fig. 3.11 implemented using ring oscillators.

design based on phase logic.

### 3.2.3 Differences from Previous Work

#### 3.2.3.1 Comparison between PHLOGON and Other Oscillator-based Computation Schemes

In comparison with other computing systems based on coupled oscillator networks [34–54] outlined in Sec. 1.2, we note that PHLOGON systems have several important differences and advantages:

- First, PHLOGON is suitable for general-purpose computation while others are only applicable to some image processing tasks, mainly including pattern recognition, image segmentation and edge detection.

- Second, PHLOGON implements Boolean computation, unlike the other ideas. Comparing PHLOGON systems with coupled-oscillator-based associative memory arrays is analogous to the comparison between digital and analog computers. Just like in digital computers, the Boolean operations in PHLOGON can tolerate considerable amount of noise and variability without resulting in any error, while the error rate in oscillator-based associative memory arrays rises markedly in the presence of noise [43]. Also, digital operations in PHLOGON systems can be coordinated with clock signals. In comparison, because of the lack of coordination, coupled-oscillator-based analog computers can reveal chaotic and unpredictable behaviors more readily. Therefore, just like regular digital computers, PHLOGON systems can more easily achieve high accuracy and reliability.

- PHLOGON allows flexible and complex system designs. It changes the signal encoding scheme and the underlying devices/circuits, but does not change the higher-level design, hence can leverage decades of design knowledge and fit easily into existing digital design flows. In comparison, available system structures for the other oscillator-based

computation schemes are much more limited.

- Non-Boolean associative computation schemes only use oscillators as one layer in the computation system. At both the input and output of this layer, it is unavoidable to convert between voltage levels and oscillation-related properties such as phase and frequency. For this purpose, controlled sources, amplifiers, comparators and even PLLs are added to each or every several oscillator cells [34, 35, 43, 45]. As nano-oscillators scale to smaller sizes, these I/O circuitries quickly dominate both area and power consumption. In PHLOGON systems, computation can proceed entirely in phase-based logic; conversion to voltage levels is not needed unless communicating to external modules [6]. We can thus reduce the I/O circuitries and take full advantage of the scalability of nano-oscillators. Indeed, the Boolean computation capability provided by PHLOGON can be used to redesign the I/O circuitries around non-Boolean associative memory layers for data processing. All these differences and possibilities have made PHLOGON a promising alternative computation scheme to investigate.

#### 3.2.3.2 Comparison of PHLOGON with Goto/von Neumann's AC-pumped Schemes

The oscillator-latch-based phase logic encoding and computing scheme described above retains the positive features of Goto's and von Neumann's original schemes. However, it also features a number of significant advantages over them:

- One of the biggest advantages of using oscillators is that it opens the door to a very wide variety of devices and systems for use as phase logic elements. Amongst molecular-sized nano-devices, spin-torque [83, 84] and tunnelling phase logic (TPL) devices [85–88] are obvious candidates. Oscillators in traditional circuit technologies are also prime candidates, given the combination of an extensive, well-established design infrastructure that already exists, and the noise immunity (Sec. 3.4.1) and energy efficiency (Sec. 3.4.2) advantages that phase based logic offers over traditional level-based logic. A wide variety of such circuit oscillators is available, including LC oscillators, ring oscillators and relaxation oscillators; their differing characteristics can potentially be mixed and matched to realize system-level advantages. Other promising oscillator candidates include electro-mechanical ones involving MEMS/NEMS elements [89]; optical oscillators (lasers), particularly miniaturizable ones such as VCSELs and Si-integrated lasers; and synthetic biological oscillators such as Elowitz's repressilator [90]. The wide variety of oscillator substrates available to implement the basic oscillator latch provides many possibilities for circumventing the disadvantages that have hindered practical adoption of phase-based logic to date.

- Unlike the circuits proposed by Goto and von Neumann (Fig. 3.4(a) and Fig. 3.1(a)), latches based on self-sustaining oscillator do not need an AC pump for power, which can

[6] Even in this scenario, we argue that we may not need to convert phase logic to voltage levels if Phase-shift Keying (PSK) is used in communication.

lead to significant system-level power/energy advantages, as discussed further in Sec. 3.4.2. Note that supplying DC power to the oscillators does not suffer from parasitic-related losses, and that the SYNC signal does not supply operating power and can be very weak, thanks to the inherent robustness of injection locking. Moreover, even unintended parasitic coupling of the SYNC signal works in favour of the scheme, since SYNC is a global signal that needs to be transmitted to every oscillator latch.

- Every member of von Neumann's latch-rings (Fig. 3.2(b)) is turned off periodically, thereby dissipating all the energy built up when it turns on. This waste of energy is avoided completely when using oscillator latches, which remain on. Oscillators can run continuously while dissipating very little energy, as described further in Sec. 3.4.2. Futhermore, having to turn von Neumann's latch-rings on periodically slows down the speed of logical operations significantly, since the circuits undergo turn-on transients that take many cycles to stabilize oscillatory waveforms. In contrast, the oscillator latches, being always on, do not need to undergo periodic turn-on transients and — thanks to the fast dynamics of injection locking — can respond very quickly to acquire phase and switch logic state.

- von Neumann's latch-ring (Fig. 3.2(b)), the basic unit for storing a logical value, requires more than one copy of the AC-pumped circuit in Fig. 3.1(a). In contrast, only one oscillator is needed for an oscillator latch — in principle, it can be as small as a single molecular-sized nano-device.

## 3.3 System Designs and Prototypes

Practically implementing PHLOGON systems relies on the modelling and simulation of oscillators, phase-macromodel-based analyses (particularly those based on GAE for predicting injection locking properties), and circuit/system-level design. We illustrate these principles and procedures with examples, using CMOS ring oscillators (Sec. 3.3.1), CMOS LC oscillators (Sec. 3.3.2), and mechanical metronomes (Sec. 3.3.3).

As mentioned above in Sec. 3.2.2, practical latches (*e.g.* a D latch) can be implemented in two ways, differing in the way the logic input (*e.g.* the D input) is controlled:

1. Switch-gated D latch, with the help of a voltage-level-controlled transmission gate;
2. Phase-controlled D latch, with the help of phase-based MAJORITY gates.

We illustrate both designs using ring oscillators in Sec. 3.3.1. Both have the same underlying mechanism and rely on the GAE analysis shown in Fig. 3.7.

Similarly, there are multiple ways phase-based logic gates can be designed:

1. Amplifier-based logic gates. Just as digital level-based logic gates are special amplifier circuits designed to accept and generate voltage levels based on inputs, an amplifier-based inverter can implement phase-based NOT operation whereas an amplifier-based three-input adder can implement phase-based MAJORITY operation. Such designs are straightforward; we illustrate them in Sec. 3.3.1.

2. Oscillator-based logic gates. Sub-harmonically injection locked oscillators can be used not just as latches, but also logic gates. We discuss this design and its merits using LC oscillators in Sec. 3.3.2.

Both schemes follow our illustration shown in Fig. 3.9 discussed in Sec. 3.2.2; they are different incarnations of the same mechanism.

### 3.3.1 PHLOGON Systems with CMOS Ring Oscillators

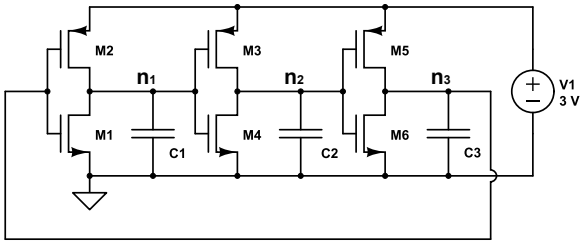



**Fig. 3.14:** Circuit schematic of a 3-stage ring oscillator.

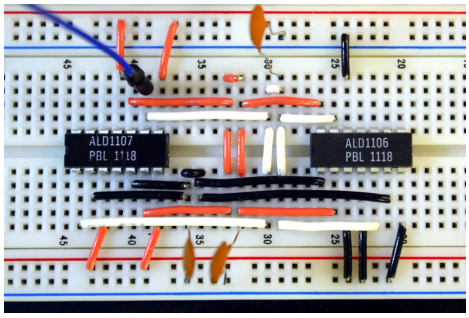



**Fig. 3.15:** Photo of a 3-stage ring oscillator on breadboard.

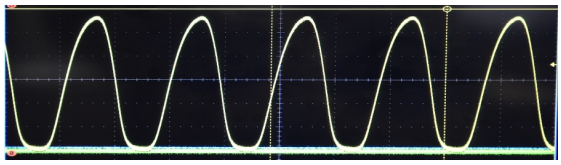


**Fig. 3.16:** Results seen from an oscilloscope.

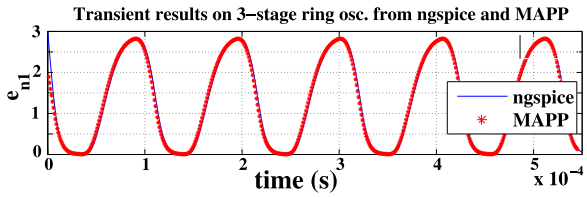



**Fig. 3.17:** Simulation results from ngspice and MAPP.

A simple ring oscillator can be made using 3 stages of CMOS inverters, each having one NMOS device and one PMOS device. Fig. 3.14 shows the circuit schematic; Fig. 3.15 shows the 3-stage ring oscillator implemented on a breadboard, with ALD1106 IC for providing the NMOS devices, ALD1107 for the PMOS. We also attach a capacitor ($C_1$, $C_1$, $C_1$ in Fig. 3.14) of 4.7nF, to slow the oscillator down for easier measurement. With a single-ended DC supply of 3V, the oscillator's response is measured in an oscilloscope in Fig. 3.16 Using both ngspice (a widely used open-source circuit simulator) and MAPP (the design tools we have developed, detailed in Chapter 5), we are able to simulate the oscillator's transient reponse

accurately (shown in Fig. 3.17). The fact that the responses from breadboard, ngspice and MAPP match well is the foundation for the following analyses and designs.

#### 3.3.1.1 Different Designs of Ring-Oscillator-based Latches

Ring oscillator D Latch with phase logic D and level logic EN

To turn a 3-stage CMOS ring oscillator into a phase-based logic latch, we need to add several inputs to it. The first one is the SYNC signal for inducing bistability in phase, which is current source attached to one of the oscillator's stages, *e.g.*, $n_1$. Then we add a fundamental frequency input signal D, connected input the same port $n_1$ through a voltage-controlled switch (VCS). The VCS is controlled by the enable signal EN. The diagram for this oscillator latch is shown in Fig. 3.18. In this straightforward design, the logic value of EN controls the on-off state of the switch, which determines if signal D will have an influence on the logic value held in the oscillator latch or not.

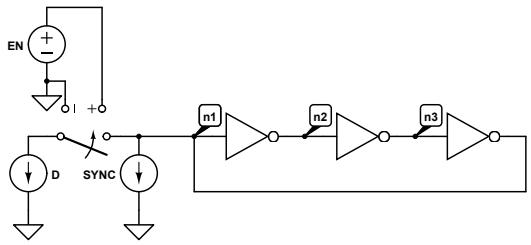



**Fig. 3.18:** Diagram of D latch made from a 3-stage ring oscillator.

For the design to achieve the functionality of a D latch, the magnitudes of signals D and SYNC need to be chosen properly. To do so, we need our design tools to extract the PPV of the oscillator, analyze its SHIL and IL properties, then determine design parameters.

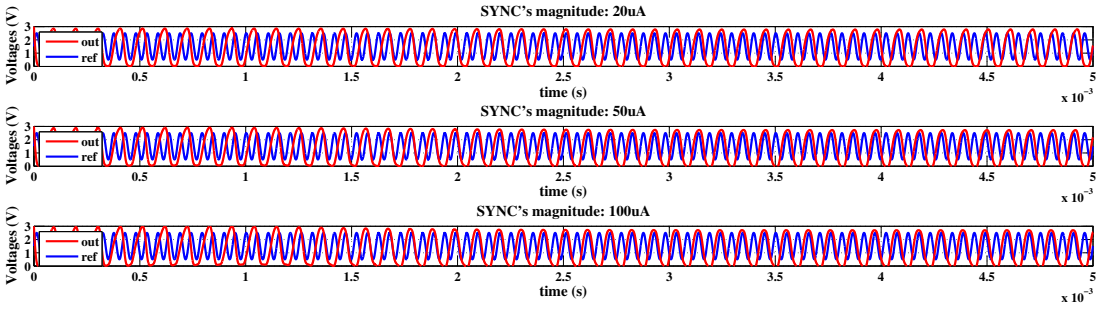



**Fig. 3.19:** Transient responses of the 3-stage ring oscillator with different SYNC magnitudes: 20$\mu$A, 50$\mu$A, 100$\mu$A.

But before that, we would like to show what designers would traditionally do for the design. For example, to determine the amount of current needed for SYNC, brute-force

transient simulations can be used. Typical results gained from these simulations are shown in Fig. 3.19; the three subplots show the transient responses of the ring oscillator with a sinusoidal SYNC with magnitudes 20$\mu$A, 50$\mu$A, and 100$\mu$A.

From the transient results, we observe that with a 20$\mu$A SYNC signal, SHIL doesn't happen; with 50$\mu$A and 100$\mu$A, it seems that SHIL occurs, but long simulations and careful detection (eyeballing or post-simulation processing) are needed before we can come to any conclusion. To determine the values of more than one such parameters, such trial-and-error method can be computationally expensive and time consuming. Moreover, the method itself doesn't provide much insight into the design or improvement of the circuit.

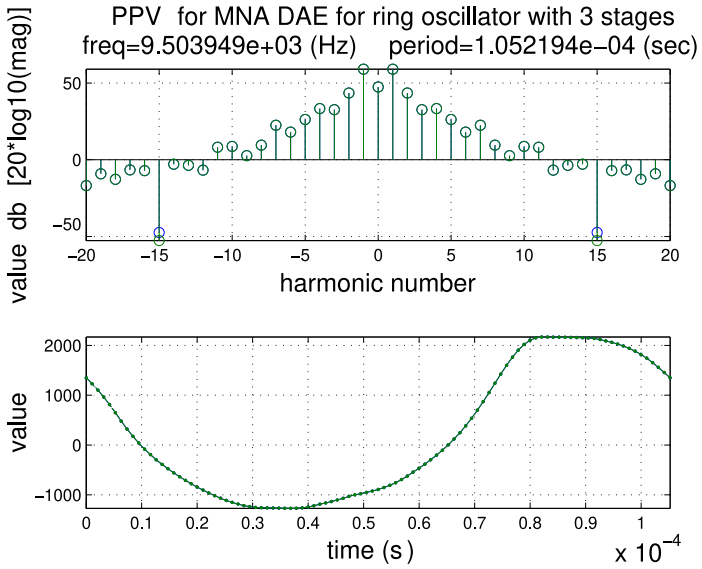



**Fig. 3.20:** First entry $\vec{v}_1(t)$ of the PPV of the 3-stage ring oscillator with SYNC.

Instead, we apply the phase-macromodel-based analysis and first extract the PPV of the free-running 3-stage ring oscillator using MAPP. Fig. 3.20 shows the shape and the frequency-domain components of the PPV entry corresponding to the current input at node $n_1$. It directly shows several quantities of the oscillator: the natural frequency is about $f_0 = 9.504$kHz; period is about $T_0 = 105.2\mu$s; the frequency component of the PPV entry at the fundamental frequency is about 57dB; the component at the second harmonic is about 45dB.

When EN=0, we would like SYNC to be large enough to induce SHIL reliably. Suppose we operate the system at $f^* = 9.6$kHz. That means SYNC should be at $f_1 = 2f^* = 19.2$kHz, In this case, the left-hand side of the GAE equilibrium equation (3.1) is

$$\frac{f_1 - 2f_0}{2f_0} \approx 0.0104. \tag{3.2}$$

It is the frequency detuning factor; the correlation function of the PPV and the SYNC signal needs to have a magnitude larger than 0.0104 for SHIL to occur. As the second harmonic

component of the PPV is $\sim$ 45dB, we can calculate the requried magnitude of SYNC to be $0.0104/10^{(45/20)} \approx 58\mu$A. This threshold value is very consistent with our observations from multiple transient simulations in Fig. 3.19. For SHIL to happen reliably, we choose a value above this threshold; the magnitude of SYNC is chosen to be $150\mu$A for this design. And this SYNC signal provides an equivalent value of $10^{(45/20)} \times 150 \times 10^{-6} \approx 0.0267$ for the A term in (3.1).

When EN=1, we would like signal D to overcome the bistability provided by SYNC and reliably set the logic value of the latch. We choose a $B$ term as large as $A$. Based on the PPV's fundamental frequency component in Fig. 3.20 at $\sim$ 58dB, we can calculate the corresponding magnitude of a sinusoidal D singal as $0.0267/10^{(58/20)} \approx 34\mu$A. Therefore, in the design, we can choose a D signal of $50\mu$A.

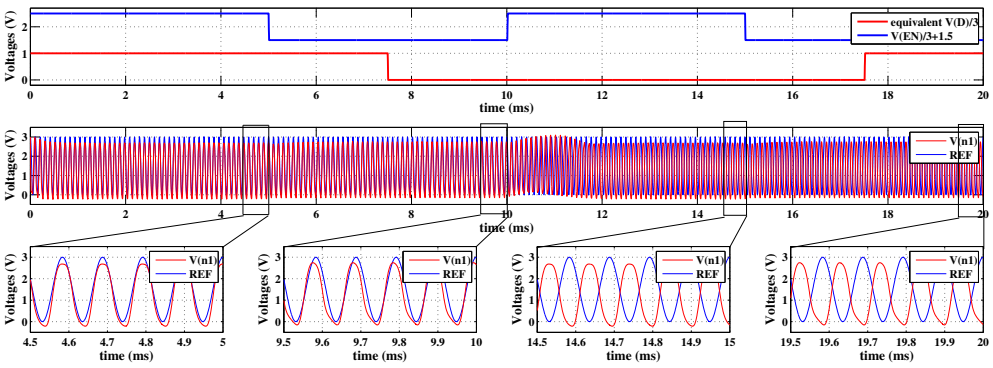



**Fig. 3.21:** Transient simulation results showing the operation of the D latch in Fig. 3.18.

From above, we have determined the design parameters: $f^* = 9.6$kHz; SYNC signal with a magnitude of $150\mu$A; D signal with a magnitude of $50\mu$A. Note that the choices are made entirely based on phase-domain analyses; expensive transient simulations are avoided.

We test the design in two ways: 1) running transient simulation for the full system and operate the designed oscillator latch with different input D/EN combinations; 2) actually building the latch on a breadboard, demonstrating its operation in hardware.

Fig. 3.21 shows the transient simulation results. From Fig. 3.21 we conclude that the operations of bit storage and flipping are achieved with this D latch. The top subplot shows the inputs: EN is a voltage-level; the D signal is an oscillatory signal whose phase is plotted instead for better visualization. We operate the latch for two EN cycles, turning it on and off in each one. In the first one, when EN=1, we use D to set the phase logic value of the latch to be 1; when EN=0, the latch's value stays at 1, even though the phase of D has switched to represent 0. Similarly, in the next cycle, when EN=1, the oscillator latches to the value of D=0; when EN=0, it holds on to 0, even though D has not shifted to 1. The four subplots are zoomed-in views of the transient waveforms. They confirm the above description and verifies that the parameters we have chosen indeed make the oscillator circuit in Fig. 3.18

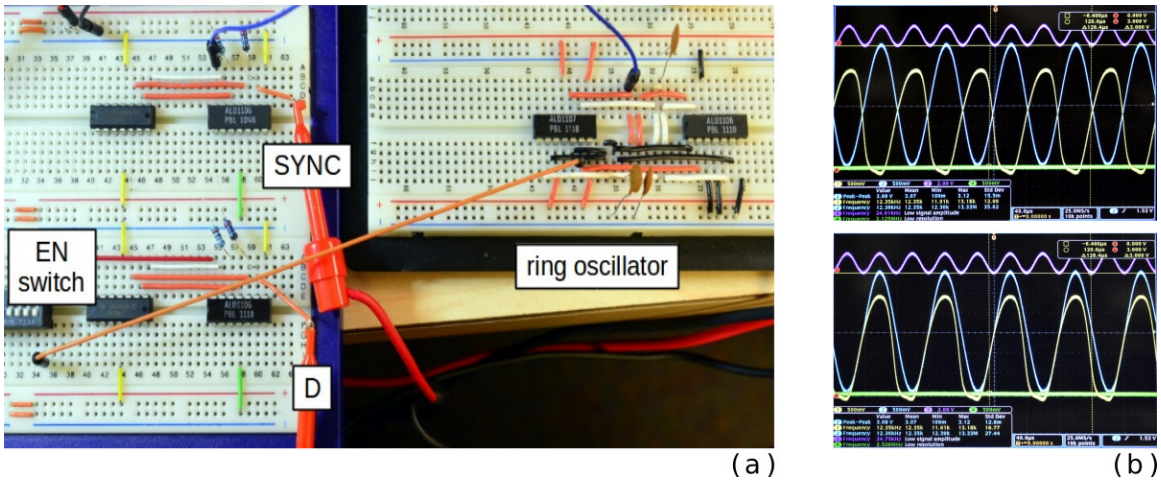
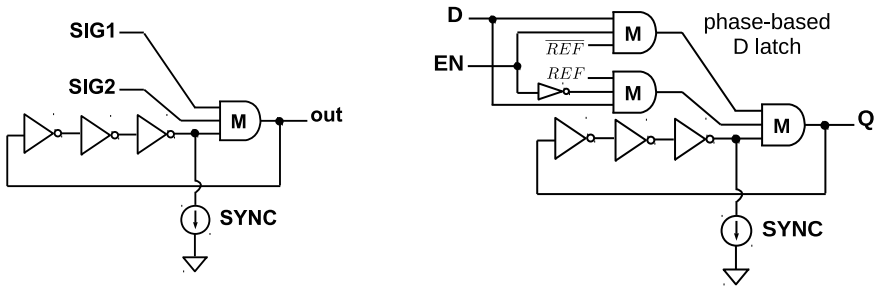



**Fig. 3.22:** Breadboard implementation of the D latch in Fig. 3.18. (a) breadboard photo; (b) waveforms showing the two logic states (yellow) in the D latch when EN=0. REF is blue; SYNC is pink.

function properly as a D latch.

Next, we build the latch on breadboard (Fig. 3.22 (a)) and observe its operation. The swith in Fig. 3.18 is implemented using a manually controlled on-off switch. From testing the implementation, we observe that the phase of the latch's output will follow D when the switch is on; it will retain its logic value when switch is off, no matter how D changes in the meanwhile. Fig. 3.22 (b) shows the two stable stored phases when the switch is off and the change of signal D has virtually no impact to the stored phases. With more testing, we conclude that the implementation as in Fig. 3.22 (a) realizes the functionality of a phase-based logic transparent D latch.

Completely Phase-based Ring Oscillator SR and D Latches




(a) Phase logic SR latch. (b) Phase logic D latch.

**Fig. 3.23:** Diagrams of phase-based logic latches.

Here, we show designs of ring-oscillator-based logic latches where all signals are oscillatory voltages with logic values encoded in their phases.

We start with a three-stage ring oscillator and design a latch with two logic inputs SIG1 and SIG2 (Fig. 3.23(a)); the latch has the functionality similar to a standard SR latch. Only when SIG1 and SIG2 have the same phase-based logic value do they have an influence on the logic value stored in the latch. Building on the SR latch, we design a D latch as in Fig. 3.23(b).

The circuit schematic of the SR latch is shown in Fig. 3.24. The main parameters to be determined are the input and feedback resistances of the majority gate, such as R1, R2, R3, R0 and Rf in Fig. 3.24. As a start, we choose R1=R2=R3=330$k\Omega$, Rf=2$\times$R0=200$k\Omega$.

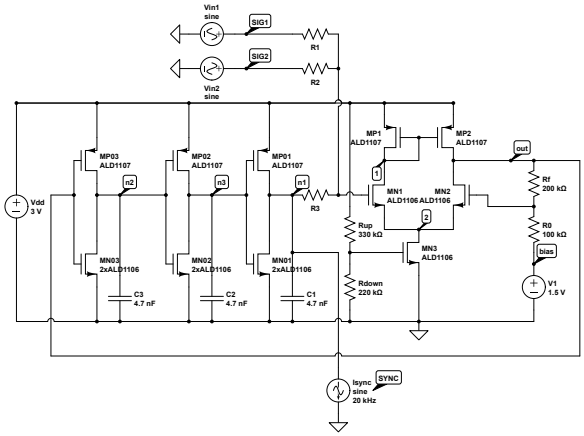



**Fig. 3.24:** Circuit schematic of the phase logic latch in Fig. 3.23(a).

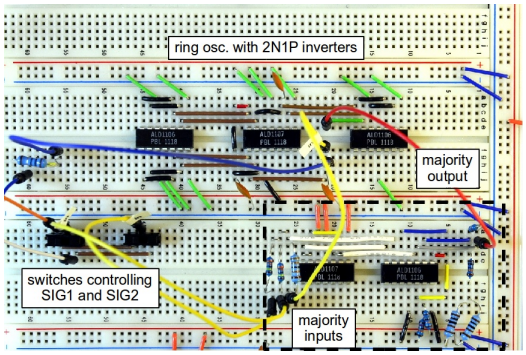
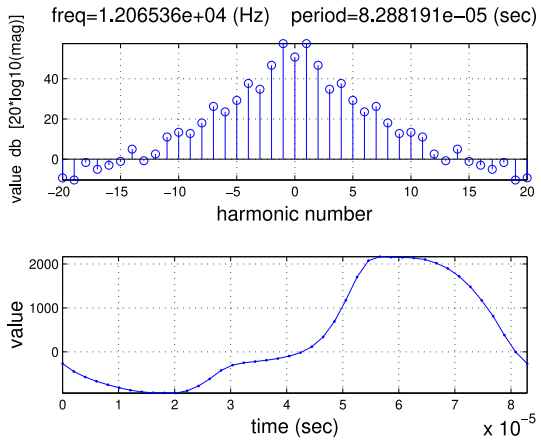



**Fig. 3.25:** Breadboard implementation of the phase logic latch in Fig. 3.24.

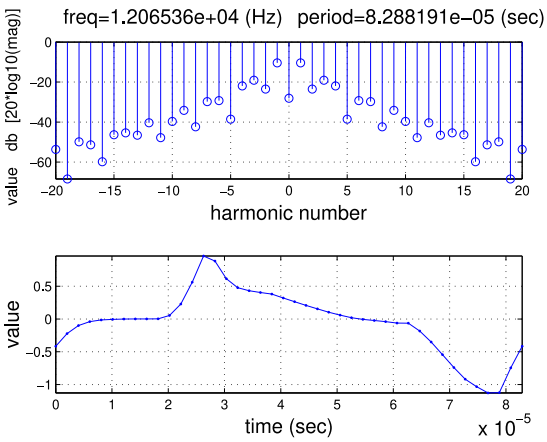
With the input ports and parameters determined, again, we extract the oscillator's PPVs corresponding to these ports. We look at the PPV entries at the KCL of $n_1$ and KVL for source Vin1 and plot them in Fig. 3.26(a) and Fig. 3.26(b) respectively, as they are the entries that are multiplied with SIG1 and SYNC in $\vec{b}(t)$.

From Fig. 3.26(a) and Fig. 3.26(b) we observe that the second-order Fourier component at the KCL_$n_1$ entry is approximately 46.7dB; the first-order Fourier component at KVL_Vin1 entry is around -10.4dB. Their ratio is about 51.7dB. This indicates that if SYNC has an magnitude of 100$\mu$A, its effect in injection locking the oscillator can be thought of as equivalent to SIG1 with magnitude of 70mV. Since SIG1 and SIG2 are symmetric in the design, this indicates when both of them have the same phase logic, once their magnitudes add up to 70mV, they will flip the bit stored in the latch; on the other hand, if they encode 1 and 0 separately and the bit should not be flipped, once they mismatch each other by 70mV at the fundamental frequency, the bit is not latched reliably any more.

With this insight from PPV analysis, we can adjust the design parameters such that the threshold for SIG1/ SIG2 to injection lock the oscillator is higher, enabling better resistance to variability and first-order perturbation. In particular, we adjust input resistors of SIG1 and SIG2 from 330k$\Omega$ to 3.3M$\Omega$. After the adjustment, the threshold increases from 70mV to approximately 700mV with the same 100$\mu$A SYNC.




(a) Entry of PPV that corresponds to input SYNC.




(b) Entry of PPV that corresponds to input SIG1.

**Fig. 3.26:** Entries of PPV for SYNC and SIG1 inputs of the oscillator latch in Fig. 3.24.

Similarly, we build the D latch as in Fig. 3.23(b) and verify its operation with various inputs and show measurements from an oscilloscope in Fig. 3.27: when EN=1, output follows D's phase logic 0 (first figure); then EN switches to 0, the phase logic 0 is stored even when D has changed to 1 (second figure); when EN=1 again, output follows D's phase logic 1 (third figure); then EN switches to 0 again, 1 is stored even when D changes to 0 (last figure). Therefore, the phase logic latch performs bit storage and bit flipping properly and achieves the functionality of a D latch.

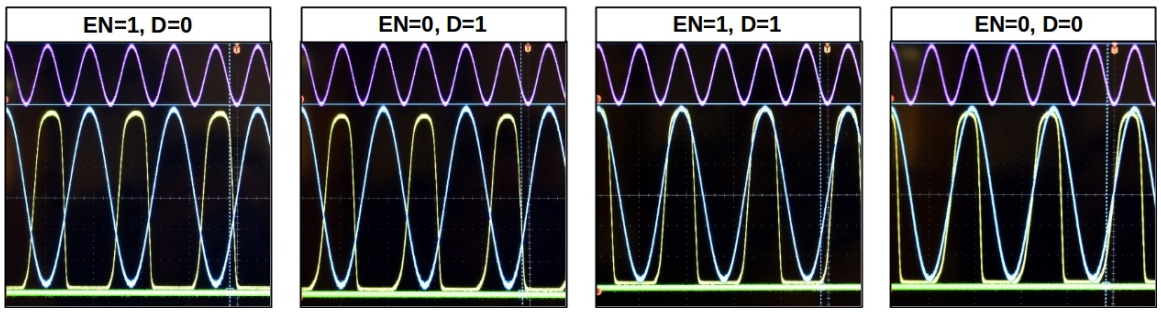



**Fig. 3.27:** Test results of the phase logic D latch seen from an oscilloscope.

### 3.3.1.2 Serial Adder: a Phase-encoded FSM Hardware Prototype

In this section, we build on the implementations of the latches to demonsrate a proof-of-concept oscillator-based FSM. We prototype a serial adder, which is a 1-bit FSM. Based on the diagram shown in Fig. 3.11, we implement the circuit on breadboard, as is shown in Fig. 3.28.

The logic gates are based on operational amplifiers (op-amps) with resistive feedback, implementing inversion for NOT gates, and addition for MAJORITY gates. Each gate uses an op-amp, provided by AD8301/AD8302 op-amp ICs. The full adder (the combinational logic block for the serial adder) consists of three MAJORITY gates and two NOT gates (as shown in the diagram Fig. 3.11). All of them are in the yellow block in Fig. 3.28. The two green blocks in Fig. 3.28 show the two D latches; they use the same design as in diagram Fig. 3.23(b).

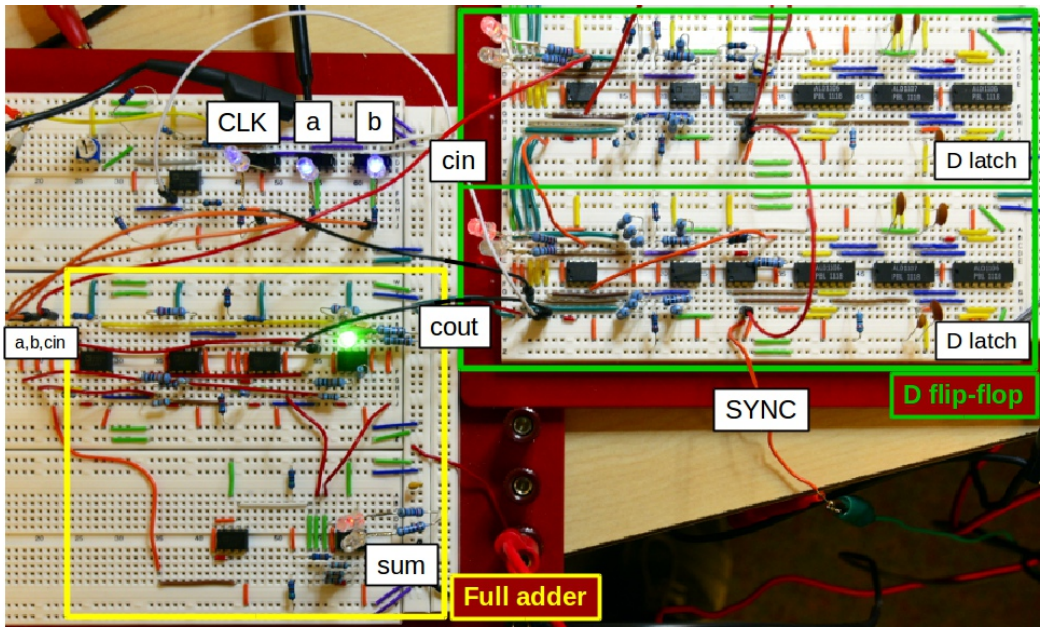



**Fig. 3.28:** Breadboard implementation of the serial adder in Fig. 3.11. The green blocks highlight the flip-flop made from two D latches; the yellow block is the combinational logic block that has the functionality of a full adder.

We first test the functionality of the D flip-flop, which is the 1-bit register in the FSM. With the input to the D flip-flop encoding different logic values, we switch CLK's phase logic value between 0 and 1 and observe the flip-flop's outputs at the two stages using an oscilloscope. The measured responses are show in Fig. 3.29. In the first subfigure, CLK=1, the master latch holds its phase logic value 0 even though input D encodes phase logic 1. Meanwhile, the slave latch follows the master with their signals overlapping on the oscilloscope; then CLK is switched to 0, the master latch's value follows input into 1 while the slave latch holds it previous value 0; then CLK is switched to 1, the slave follows the master into 1; then D is changed to 0 and another CLK cycle passed. Similarly, the master flips to 0 with the falling edge of CLK's phase, while the slave follows to 0 with the following rising edge. The results confirm the logic function of the master-slave D flip-flop.

Then we connect the D flip-flop with the combinational logic block implemented with majority and not gates. We test the operation of the resulting serial adder with various

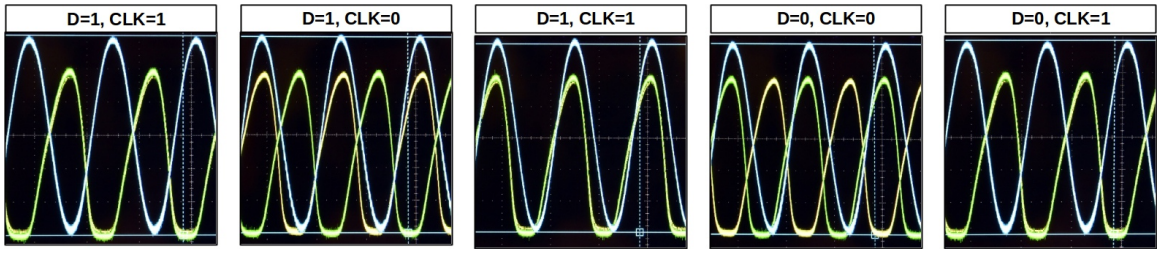



**Fig. 3.29:** Test results of the master-slave D flip-flop as shown in the green block of Fig. 3.28. REF, output of master, output of slave are displayed with blue, green, yellow respectively.

input sequences and confirm that the circuit is indeed a finite state machine with 1-bit state operating completely using phase-based logic. Selective results observed from the oscilloscope are shown in Fig. 3.30. In the two subfigures in Fig. 3.30 the inputs a, b are the same: a=0 and b=1. The left subfigure shows sum=1, cout=0 when the state machine is at carry=0 state; the right one shows sum=0, cout=1 when the state machine is at carry=1 state.

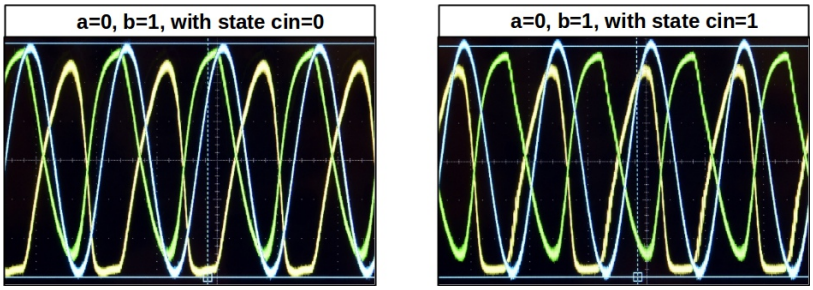



**Fig. 3.30:** Selected test results of the serial adder as in Fig. 3.28. REF, sum, cout are displayed with blue, green, yellow respectively.

### 3.3.2 PHLOGON Systems with CMOS LC Oscillators

In this section, we explore the use of CMOS LC oscillators for PHLOGON systems. Fig. 3.31 shows the schematic of a cross-coupled CMOS LC oscillator. It is a natural choice for implementing a binary phase-based latch as its PPV has an entry consisting almost entirely of second harmonic component (Fig. 3.32).

Similar to our exploration with ring oscillators, we have built these LC oscillators on breadboards, using ALD1106/ALD1107 CMOS devices, 10m inductors, 6.8nF capacitors for a central frequency around 10kHz. With a SYNC input at 20kHz, we observe that such cross-coupled LC oscillators feature SHIL prominently. Unlike the design with ring oscillators, the signal SYNC does not have to be a current any more; it is a voltage directly applied at the gate of the source NMOS, as is shown in Fig. 3.31. This simplifies the system layout significantly. Also, the waveforms are sinusoidal with little distortion, indicating good energy efficiency in sustaining the oscillation. When SYNC is attached, there is virtually

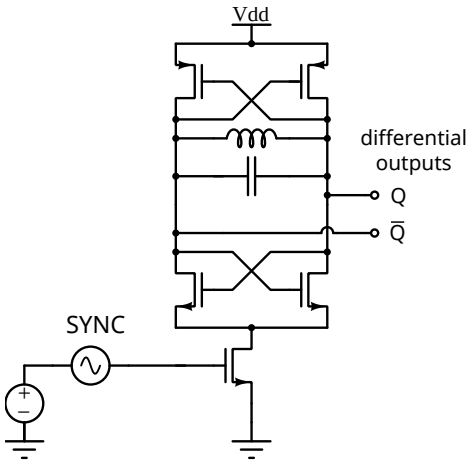



**Fig. 3.31:** Schematic of a complementary cross-coupled LC oscillator.

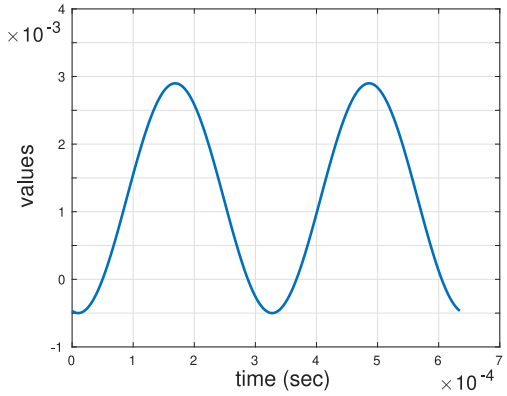
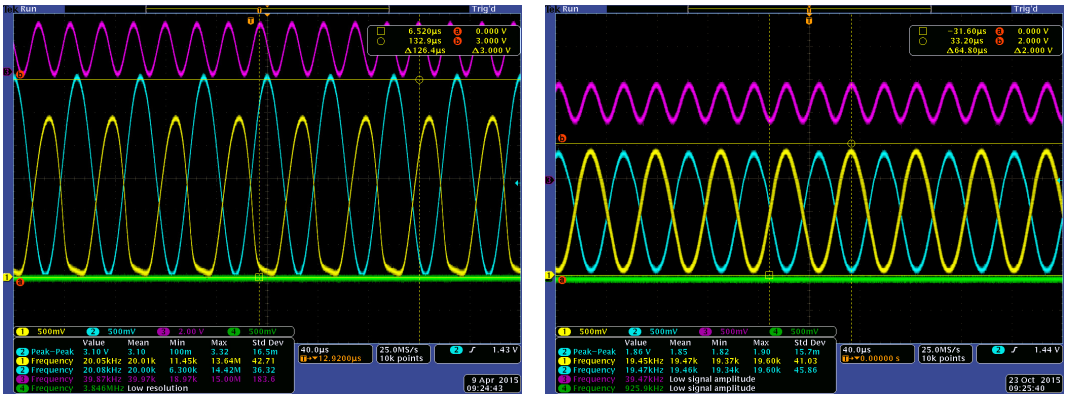



**Fig. 3.32:** PPV entry of the cross-coupled LC oscillator for the SYNC input.

no change in the waveform; it is mainly used to stabilize and binarize the phase instead of supplying power.



(a) Waveform from ring oscillator under SHIL. (b) Waveform from LC oscillator under SHIL.

**Fig. 3.33:** Comparison between waveforms from ring-oscillator- and LC-oscillator-based PHLOGON systems. SYNC is in pink; REF is in blue; signals from oscillator latches are in yellow.

The design of the logic gates have also been mosified with the help of the LC oscillators. Since cross-coupled LC oscillators are differential, one latch can supply both the stored bit and its inverse, *i.e.*, $Q$ and $\overline{Q}$. Such an differential oscillator can not only be used as a latch, it is also directly a NOT gate. Similarly, MAJORITY gates can also be implemented using oscillators, simply by letting three inputs perturb the oscillator at the same time. When the oscillator is injection locked by the inputs and settle to a stable phase, it will pick the majority of the input phases. In this way, both latches and logic gates are implemented using oscillators; the resulting PHLOGON system is akin to coupled oscillator network.

Such an oscillator-based realization of phase-encoded logic gates has several advantages compared with the op-amp-based implementation.

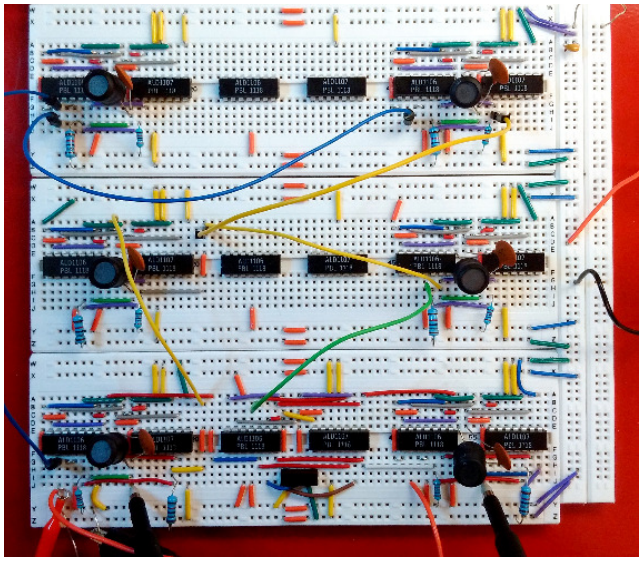


**Fig. 3.34:** Photo of breadboard circuit of an LC-oscillator-based PHLOGON system.

- The outputs from the gates are less distorted, and the magnitude of output is corrected naturally without the use of voltage limiter (which introduces distortion) or Automatic Gain Control (AGC) circuits.

- The system consists of a group of self-sustaining oscillators without op-amps. It is conceptually cleaner. This feature is very attractive for the practical implementation of PHLOGON in physical domains other than electronics (*e.g.*, coupled spin-torque oscillators or lasers) where constructs equivalent to the op-amp-based circuitry may not be available.

- The phase error caused by the delay from the gate can be "corrected". In this case, SYNC can be attached to both the oscillator latches and the oscillator logic gates; it will try to pull the output phase to align with logic 1 or 0 in both scenarios. This is analogous to level-based logic gates in CMOS, where imperfect inputs are corrected to be Vdd or gnd at output.

Based on the new design, we have prototyped another full adder using cross-coupled CMOS LC oscillators on a breadboard (Fig. 3.34). We are able to verify its operation as a FSM through measurements on a oscilloscope.

### 3.3.3 Achieving SHIL and Bit Storage in Mechanical Metronomes

The mechanism for oscillators to be used as the substrate for Boolean computation is not specific to electrical CMOS oscillators. Indeed, as mentioned above in this chapter, a wide

range of oscillators from multiple physical domains are suitable for PHLOGON. In this section, we use a common type of mechanical oscillators — metronomes as an example, and use them to make phase-based logic latches [91, 92].

The synchronization of metronomes is perhaps the most famous example of injection locking in oscillators. As illustrated in Fig. 3.35, when several metronomes are placed on a platform that can roll horizontally, each of them receives a small perturbation from their neighbours through the common platform. They may have slightly different central frequencies and may be started with random phases. But given some time, through the mechanism of injection locking, all of them will eventually lock to the same frequency with the same phase. This synchronization phenomenon is reproducible, easy to see and hear. As such, it is often used to illustrate or teach the subject of injection locking.

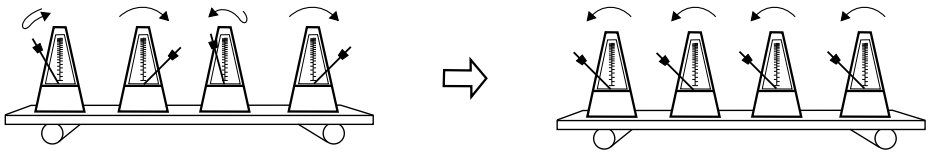


**Fig. 3.35:** Metronomes standing on a rolling board end up ticking in unison.

But to our knowledge, SHIL has never been demonstrated in mechanical metronomes. The basic idea of such a demonstration is simple: just like in the scenario of regular injection locking, or IL, two metronomes are placed on a rolling board — one oscillates at approximately the $1/2$ sub-harmonic of the other. If SHIL happens, their swing patterns will synchronize. To complete this demonstration, they should also be decoupled by stopping the rolling board, in which case the synchronization ceases. Such a demonstration can show that SHIL is not limited to electrical oscillators discussed in Sec. 3.3.1 and Sec. 3.3.2 — it is almost as universal as regular IL. It can also serve as an eye-catching illustration when teaching the subject of SHIL in classes.

The lack of such a demonstration is not because of lack of trying. In fact, we have been attempting to demonstrate it in our group for years. But simply tuning two metronomes to 1Hz and 2Hz[7] and putting them on a rolling board does not result in synchronization. The coupling seems to have little effect and the detuning in their frequencies remains, separating their phases apart. The metronomes we use[8] oscillate for approximately 17 min at 2Hz; clear and secure synchronization appears impossible to achieve within this time frame. This observation leaves us an impression that metronomes are very immune to perturbation close to the second harmonic of its natural oscillation (*aka*, second-order perturbation). This intuition, however, needs more concrete analysis and justification.

In this section, we adopt a more rigorous approach, similar to our analyses in Sec. 3.3.1

[7]Frequencies used in this section are the frequencies of metronomes' oscillation. One cycle of metronome oscillation generates two ticks. So 1Hz and 2Hz metronomes generate 120 and 240 beats per minute respectively.

[8]Wittner Taktell Super-Mini Metronome

and Sec. 3.3.2 for the two types of electrical oscillators. We start with deriving physical compact models of metronomes that that can run in open-source and commercial simulators. Metronomes are normally modelled as lossless double-weighted pendulums [93, 94]. But the "lossiness" of an oscillator actually plays an important role in its injection locking behavior. An accurate metronome model needs to include friction damping and also a mechanism known as escapement that compensates the energy lost every cycle due to friction. The implementations of the several existing metronome models with these mechanisms are not openly available. Neither are they formulated to be compatible with main-stream simulators. Moreover, friction and the escapement mechanism are often modelled in these models using non-smooth functions, making them not suitable for simulation. In our model, we use smooth functions to alleviate the difficulty of convergence in simulation, and also increase the models' physical fidelity.

Then we perform phase-based analysis on the model and explain why SHIL is hard to achieve. To our knowledge, this is the first time metronomes are analyzed using phase-macromodel-based techniques. The analysis also allows us to tweak the metronome and adjust its PPV such that SHIL can happen more easily. Building on the analysis and adjustment, we validate the occurrence of SHIL — both frequency and phase locks in metronomes by experiments and measurements. The result is a *phase-based logic latch* based on a *mechanical oscillator*.

The techniques we present in this section on the modelling and analysis of metronomes, especially the use of phase-macromodels to predict, analyze and achieve desirable IL properties, are applicable to and useful for the design of almost any oscillator. The easily reproducible results demonstrate the generality of SHIL in oscillators. Just as metronomes are often used in teaching the subject of IL, our experiments in this section can also be used for teaching the subjects of SHIL, oscillator modelling and design, as well as oscillator-based Boolean computation in classrooms.

#### 3.3.3.1 Modelling Metronomes

The dynamics of a metronome are mainly governed by the equation of a double-weighted pendulum. The equation is written using the angle $\theta$ and angular velocity $\dot{\theta} = \frac{d}{dt}\theta$ of the pendulum[9]:

$$\frac{d}{dt}\dot{\theta} = -\frac{1}{m_1 h_1^2 + m_2 h_2^2} \cdot (m_1 g \sin(\theta) \cdot h_1 - m_2 g \sin(\theta) \cdot h_2), \tag{3.3}$$

where $m_1$ is the mass of the weight at the bottom of the pendulum, $h_1$ is the distance from it to the axis of rotation; $m_2$ and $h_2$ are the mass and distance for the weight on the top of the pendulum; $g$ is the gravitational constant.

[9]The angle is defined between the pendulum and the vertical position. Changing the definition to use the other direction does not change any equation.

Next, we add several terms to this pendulum equation — friction, the escapement mechanism, and the external perturbation.

Frictional forces in this system come from several sources: air friction, the axle bearing, and the contact between the tooth of the escapement wheel and the circular plate of the actuating member attached to the axle. The first one — air friction is often modelled as a viscous friction that grows linearly with velocity. The latter two are in the form of surface friction, which is often considered as a constant force in the opposite direction of the relative movement. They are much larger than the air friction in a metronome. Therefore, we write the formula for friction as

$$f(\dot{\theta}) = -f_0 \cdot \text{smoothsign}(\dot{\theta}), \tag{3.4}$$

where $f_0$ is a fitting parameter representing the constant amplitude of friction.

A double-weighted pendulum with only frictional forces will have damped oscillation. To sustain the oscillation, a metronome has a spring box inside that drives an escapement wheel through gears. The wheel has sloped teeth. At almost any time, one of the teeth is pushing against a circular plate attached to the axle of the pendulum. As the pendulum swings to certain angles where the circular plate is designed to have a gap, the wheel "escapes" through the plate and moves one tooth forward. In the meanwhile, the movement of the tooth pushes the circular plate, making the pendulum swing faster. At the same time, a sound of a tick is generated. This escapement happens twice during a cycle of oscillation, one when the pendulum is swing left and the other right. The angles at which the escapement occurs at left and right are normally symmetric. The pushing force on the pendulum generated by escapement is modelled as a $g(\theta,\ \dot{\theta})$ function in our model, which is non-zero only at small windows of $\theta$, with direction aligned with the sign of $\dot{\theta}$.

$$\begin{aligned} g(\theta,\ \dot{\theta}) = &+ g_0 \cdot (\text{smoothstep}(\theta - \theta_R) - \text{smoothstep}(\theta - \theta_R - \Delta\theta)) \cdot \text{smoothstep}(\dot{\theta}) \\ &- g_0 \cdot (\text{smoothstep}(\theta - \theta_L + \Delta\theta) - \text{smoothstep}(\theta - \theta_L)) \cdot \text{smoothstep}(-\dot{\theta}). \end{aligned} \tag{3.5}$$

The formula in (3.5) sets $g(\theta,\ \dot{\theta})$ to be about $g_0$ when $\theta_R < \theta < \theta_R + \Delta\theta$ and $\dot{\theta} > 0$; $g(\theta,\ \dot{\theta})$ is about $-g_0$ when $\theta_L - \Delta\theta < \theta < \theta_L$ and $\dot{\theta} < 0$.

Furthermore, the external force applied to a metronome can be written as a horizontal acceleration $a$ of the axle of the pendulum.

Putting together all the components discussed above, we have a metronome model as follows.

$$\begin{aligned} \frac{d}{dt}\dot{\theta} = &- \frac{1}{m_1 h_1^2 + m_2 h_2^2} \cdot (m_1 g \sin(\theta) \cdot h_1 - m_2 g \sin(\theta) \cdot h_2 \\ &+ f(\dot{\theta}) + g(\theta,\ \dot{\theta}) + m_1 \cdot a \cdot \cos(\theta) \cdot h_1 - m_2 \cdot a \cdot \cos(\theta) \cdot h_2 \big). \end{aligned} \tag{3.6}$$

Among the parameters used in this model, $m_1$, $m_2$, $h_1$ and $h_2$ can be directly measured using a scale and a ruler; values for $\theta_R$, $\theta_L$ and $\Delta\theta$ can be obtained by measuring the gap in the circular plate using a protractor. $f_0$ is the nominal value of friction. We can estimate it by letting the pendulum swing within small angles, such that the escapement does not happen. In this case, the metronome begins damped oscillation due to friction. We can tweak the value of parameter $f_0$ until the simulated response matches observation in the speed of damping. $g_0$ is the force the escapement wheel applies to the pendulum during each tick. When all the other parameters are fixed, $g_0$ determines the swing of the metronome. We can estimate this parameter by matching the magnitude of oscillation in simulation with the actual swing of the metronome. From above, we have been able to systematically determine all the parameters in the metronome model (3.6), either directly or indirectly.

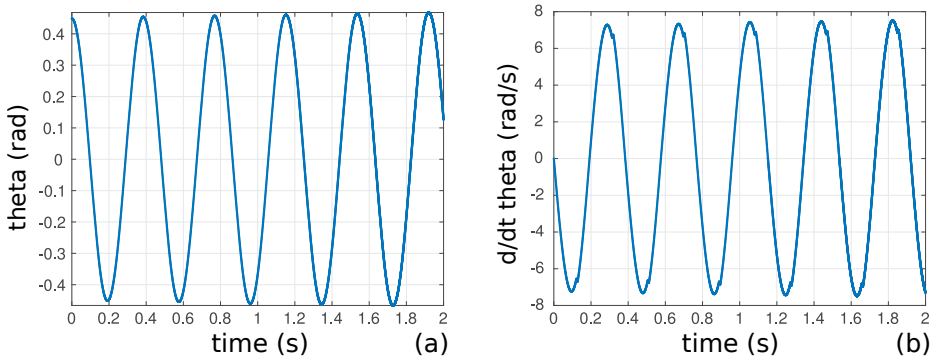



**Fig. 3.36:** Transient simulation results of $\theta$ and $\dot{\theta}$.

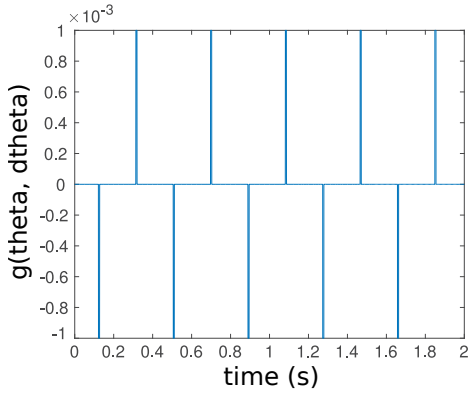



**Fig. 3.37:** Transient simulation results of $g(\theta,\ \dot{\theta})$.

Note that in the model equations for friction (3.4) and escapement mechanism (3.5), we use the smooth versions of sign and step functions [95, 96]. This is not just for improving the convergence of numerical simulation. The use of smooth functions also makes the model represent the physical system more truthfully. For example, when the tooth of the escapement wheel starts to push the circular plate, the force is not instantaneous because the

surface at the edge of the gap is still smooth. Similarly, representing friction using smooth functions also increases the model's physical fidelity [97].

Results from transient analysis in Fig. 3.36 show that the model reproduces the self-sustaining oscillation observed in metronomes. Note that the waveforms of $\theta$ and $\dot{\theta}$ are not perfectly sinusoidal, $\dot{\theta}$ actually has two small notches every cycle — they are where the escapement wheel moves forward one tooth. Plot of $g(\theta,\ \dot{\theta})$ function in Fig. 3.37 demonstrates this mechanism more clearly; unlike a spring-mass system or a single pendulum, a metronome is indeed a nonlinear oscillator with highly non-smooth dynamics.

### 3.3.3.2 Tweaking Metronomes based on Phase Domain Analysis

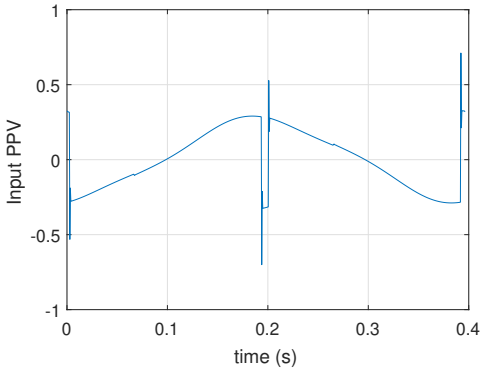



**Fig. 3.38:** Input PPV of metronome.

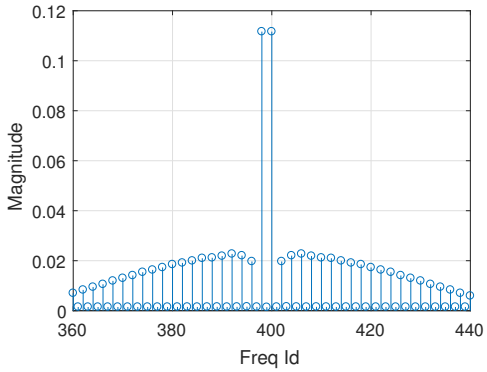



**Fig. 3.39:** Frequency domain coefficients of metronome's PPV.

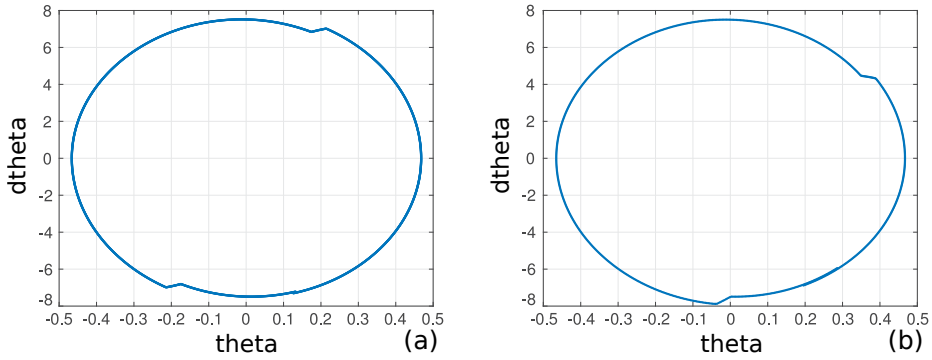



**Fig. 3.40:** Phase portrait of a metronome's oscillation before and after modification.

Fig. 3.38 and Fig. 3.39 show the time- and frequency-domain PPV of a metronome. The metronome model has only one input variable $a$, which is the horizontal external input acceleration; the PPV becomes a time-varying scaler that describes the phase's sensitivity to this input. From Fig. 3.39 we see that the PPV has a large coefficient at the fundamental frequency, indicating that a metronome is prone to regular first-order IL. But the second harmonic of the PPV is around $10^{-12}$, in the order of numerical noise. In fact, all even

harmonics are practically zero, because both the waveforms and PPV of a metronome are designed to be odd symmetric. It would be easier to SHIL the metronome with a periodic input $u(t)$ at $3f_1$, but if we insist on using $2f_1$, the metronome needs to be modified.

The phase portrait in Fig. 3.40 (a) illustrates the oscillation of a metronome; it is another way of visualizing the $\theta$ and $\dot{\theta}$ waveforms in Fig. 3.36. The two small kinks in the loop result from the escapement mechanism, which accelerates $\dot{\theta}$ for a short time. They are symmetric about the origin. If we open up the metronome, use pliers to rotate the circular plate, we can adjust the values of $\theta_R$ and $\theta_L$ in (3.5), thus making the phase portrait asymmetric, as illustrated in Fig. 3.40 (b). After this modification, the two ticks generated in each cycle are not equally separated anymore. But interestingly, the duration of the metronome oscillation does not change; the modification does not seem to affect the metronome's energy consumption.

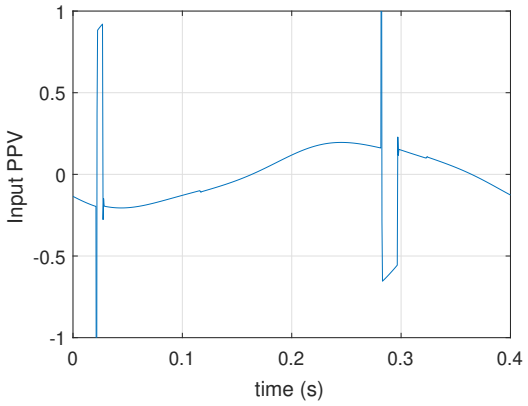



**Fig. 3.41:** Input PPV of modified metronome.

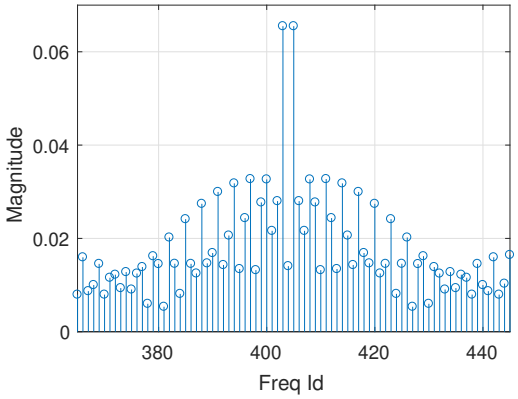



**Fig. 3.42:** Frequency domain coefficients of modified metronome's PPV.

Fig. 3.41 and Fig. 3.42 show the PPV of the metronome after modification. Compared against Fig. 3.38 and Fig. 3.39, the second harmonic of the PPV increases from zero to 0.028, which is now about 43% of the coefficient at fundamental frequency. The observation of SHIL should now becomes considerably easier; it should be almost as reproducible as regular IL in metronomes.

#### 3.3.3.3 SHIL in Metronomes

In the experiment for SHIL, we placed two metronomes on a rolling board — one tuned to around 1Hz, the other around 2Hz. The 1Hz metronome was modified according to the previous section, to increase its susceptibility to second-order perturbation. A red sticker was taped to the rod of the 1Hz metronome; a blue one to the 2Hz metronome. With a tripod, we recorded their oscillation with a 30Hz frame rate. The video was imported frame by frame into MATLAB®; an simple algorithm was used to extract the locations of red and blue stickers from the video. For every frame, we determined the centers of the two stickers

by selecting fixed numbers of red-most and blue-most pixels and averaging their locations respectively. Fig. 3.43 (d) shows the oscillation in the two stickers' x coordinates within a time frame of about 20 seconds.

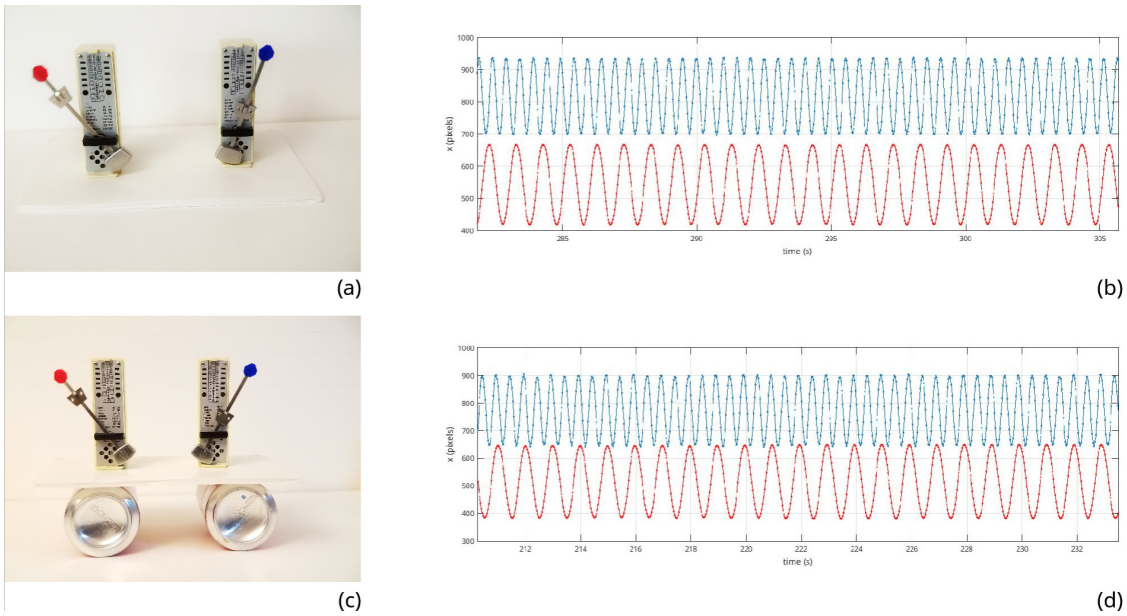



**Fig. 3.43:** (a) Two metronomes placed on a stationary board; (b) corresponding waveforms of the tips of their pendulum rods; (c) two metronomes on a rolling board; (d) their corresponding waveforms.

Similarly, without retuning the metronomes, we placed them on a stationary board instead, processed the video and plotted the oscillation in Fig. 3.43 (b). Throughout the plot, the peaks of the red waveform are almost aligned to every other valleys of the blue waveform. However, at the beginning of the plot, the red peak is on the right of the blue valley, but towards the end of the plot, it has clearly drifted to the left of the corresponding blue valley. This is an indication that the two metronomes are not synchronized, whereas in Fig. 3.43 (d), the two waveforms are aligned within the same duration, indicating the occurrence of SHIL.

The curves in Fig. 3.44 (b) and (d) are known as the Lissajous curves — they plot the x coordinates of the red dot *wrt* those of the blue dot. On a rolling board, Lissajous curves in Fig. 3.44 (d) clearly show a pattern, indicating that the two metronomes synchronize. In the case of the stationary board, because of detuning, the Lissajous curves span the whole plane within the swings of oscillation.

We can also identify all the peaks in Fig. 3.43 (b) and (d) with the help of a MATLAB®command `findpeaks()`. Based on the locations of the peaks, we can plot the phase difference between the two metronomes in Fig. 3.45. When the two metronomes are free running on a stationary board, the phase difference keeps increasing. Specifically, the 1Hz metronome lags the 2Hz one by approximately 10 cycles after 600 seconds. But when

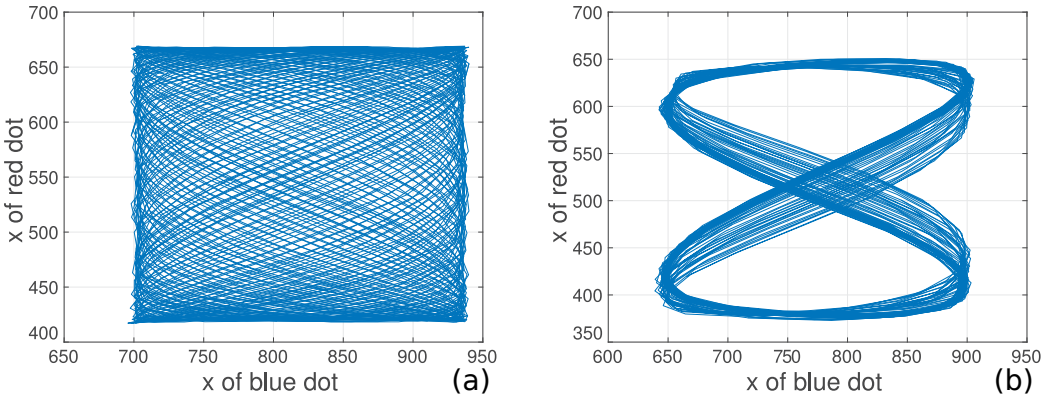



**Fig. 3.44:** (a) Lissajous curves Fig. 3.43 (b), showing that no synchronization happens; (b) Lissajous curves Fig. 3.43 (d), with the pattern demonstrating SHIL.

they are on the rolling board, the phase difference stays constant around zero for the same duration. Put in other words, every 2 cycles of the 2Hz metronome align almost perfectly with 1 cycle of the 1Hz one. The measurements are not perfectly flat mainly because the spring box does not generate a constant force, and a metronome's frequency changes slightly as the spring unrolls. Also, as the two metronomes oscillate beyond 10 minutes towards the end of their oscillation, their frequencies drift more and more, and SHIL is eventually broken.

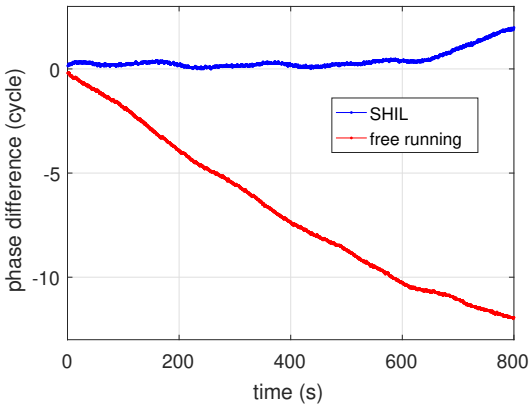



**Fig. 3.45:** Phase difference between the 2Hz and 1Hz metronomes. The difference is measured by the number of cycles of the 1Hz metronome. Positive values mean that the 1Hz metronome is leading the 2Hz one.

Furthermore, from the locations of the peaks, we can calculate the frequencies of the two metronomes. From the stationary board to the rolling board, the 2Hz metronome maintains the same frequency at $2f_1 = 1.978$, but the 1Hz one changes its frequency by about 1%, from $f_0 = 0.998$ to $f_1 = 0.989$. This indicates that it is the slower metronome that moves its frequency from $f_0$ to $f_1$, same as our expectation.

#### 3.3.3.4 Bit Storage in Metronomes

SHIL features not only frequency lock (Fig. 3.44), but also phase lock. The 1Hz oscillator in Fig. 3.43 should develop a bistable phase, allowing it to operate as a binary logic latch. However, this is not possible to observe through a setup like Fig. 3.43, as logic encoding based on phase requires a reference, *e.g.*, another metronome tuned to approximately 1Hz. In other words, we need two 1Hz metronomes, both injection locked by the same 2Hz metronome, such that their phase difference can then be used to store a logic bit.

The main difficulty with this scheme is that the 2Hz metronome needs to injection lock the two 1Hz ones "independently", *i.e.*, the two 1Hz ones should not injection lock each other. To do so, we design a metronome placement scheme different from the conventional one in Fig. 3.35. The two 1Hz metronomes are placed with a 90° angle, such that the directions in which they swing are perpendicular. To truly decouple them, the platform now needs to be able to roll freely in the horizontal plane. So instead of using cylinders as in Fig. 3.35, we use balls to support the platform on the table. The third metronome, which is tuned to 2Hz, is then placed in between the two 1Hz ones, with a 45° angle from both of them. In this way, its swing injection locks them simultaneously. The setup is illustrated in Fig. 3.46.

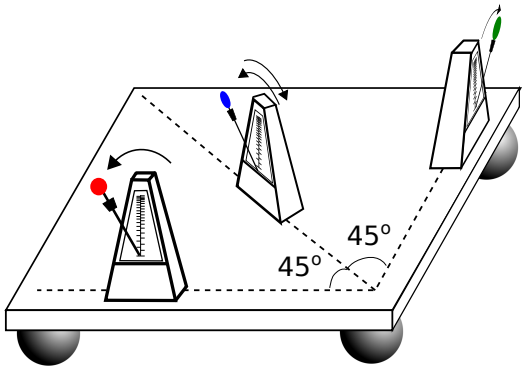



**Fig. 3.46:** Illustration of the experimental setup, where two 1Hz metronomes (with red and green tapes) oscillate in perpendicular directions, and one 2Hz metronome is placed between them with a 45° angle.

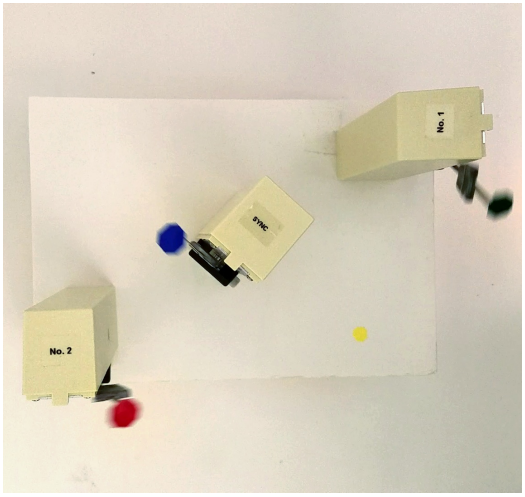



**Fig. 3.47:** Photo of the experimental setup.

There are many everyday objects that can be used as the platform and balls in Fig. 3.46. In practice, we would like to minimize the total mass of the setup in order to maximize the injection between metronomes. Therefore, we use a small foam board instead of a wooden one, and four ping-pong balls beneath it. Such details can make a difference in the reliability of the setup and the reproducibility of the results.

As shown in the photo of the actual setup in Fig. 3.47, the two 1Hz metronomes are taped with red and green markers, the 2Hz one is taped with a blue marker. We start

the two 1Hz metronomes first and confirm that their ticking are not synchronized due to frequency detuning. Then we start the 2Hz one. Through the mechanism of SHIL, the two 1Hz metronomes are both frequency locked to the 2Hz one, and they develop a stable phase difference, which stores one logic value. We can manually stop one of the 1Hz metronomes for half a cycle then let it resume its oscillation. After the two 1Hz metronomes are synchronized by the 2Hz one again, their stable phase difference is changed from before by 180°, representing the other binary logic value. We record the whole process using a cell phone with a 60Hz frame rate. The video is imported frame by frame into MATLAB®. Then a simple algorithm is used to extract the locations of markers in each frame. Fig. 3.48 shows the oscillation of the coordinates of the markers through the whole experiment of about 270 seconds.

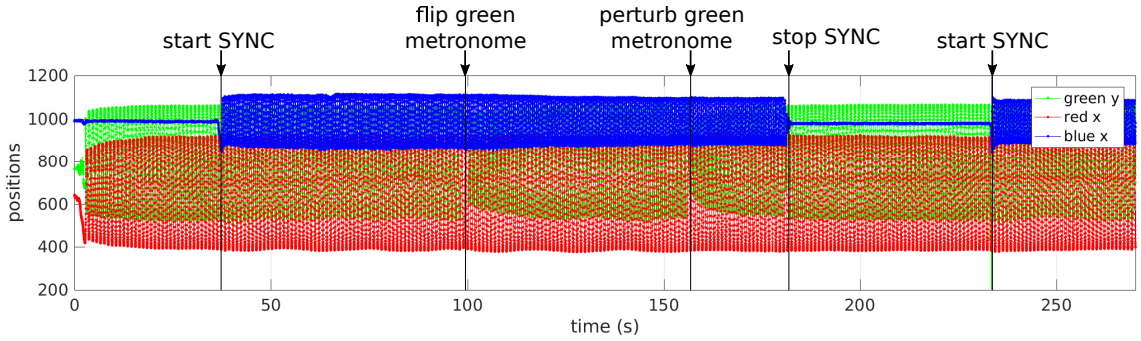



**Fig. 3.48:** X or Y coordinates of the tips of the three metronomes in Fig. 3.47 during the experiment.

In Fig. 3.49, we show some excerpts from Fig. 3.48. The first excerpt in Fig. 3.49 (a) shows the oscillation of the two 1Hz metronomes from 10s to 30s, during which time the 2Hz one has not been started. Careful observation indicates that the green and red waveforms are not synchronized — the red peak is aligned with the green valley at the beginning of the time slot but has apparently become misaligned towards the end. The corresponding Lissajous curves more clearly show that the phase difference drifts with time. In comparison, the second excerpt (Fig. 3.49 (c)) is taken between time 65s and 85s, when SHIL is present. We observe that the green and red peaks are well aligned, as can be confirmed by the corresponding Lissajous curves in Fig. 3.49 (d). Similarly, after the green metronome's phase is flipped, from time 130s to 150s, the red peaks are aligned with the green valleys (Fig. 3.49 (e)). This other stable phase difference represents the other phase-encoded logic value. In these two stable states, the corresponding Lissajous curves in Fig. 3.49 (d) and (f) both form a line, but the two lines are perpendicular to each other.

To explore the stability of the stored bit, we perturb the green metronome at about 155s in the experiment, as is shown in Fig. 3.48. Unlike bit flipping, we touch the metronome to slightly delay its oscillation. As a result, the two 1Hz metronomes do not settle to the new phase difference; their previous stable phase difference is restored within ten cycles. This further validates that the setup is a bistable system storing a phase-encoded bit.

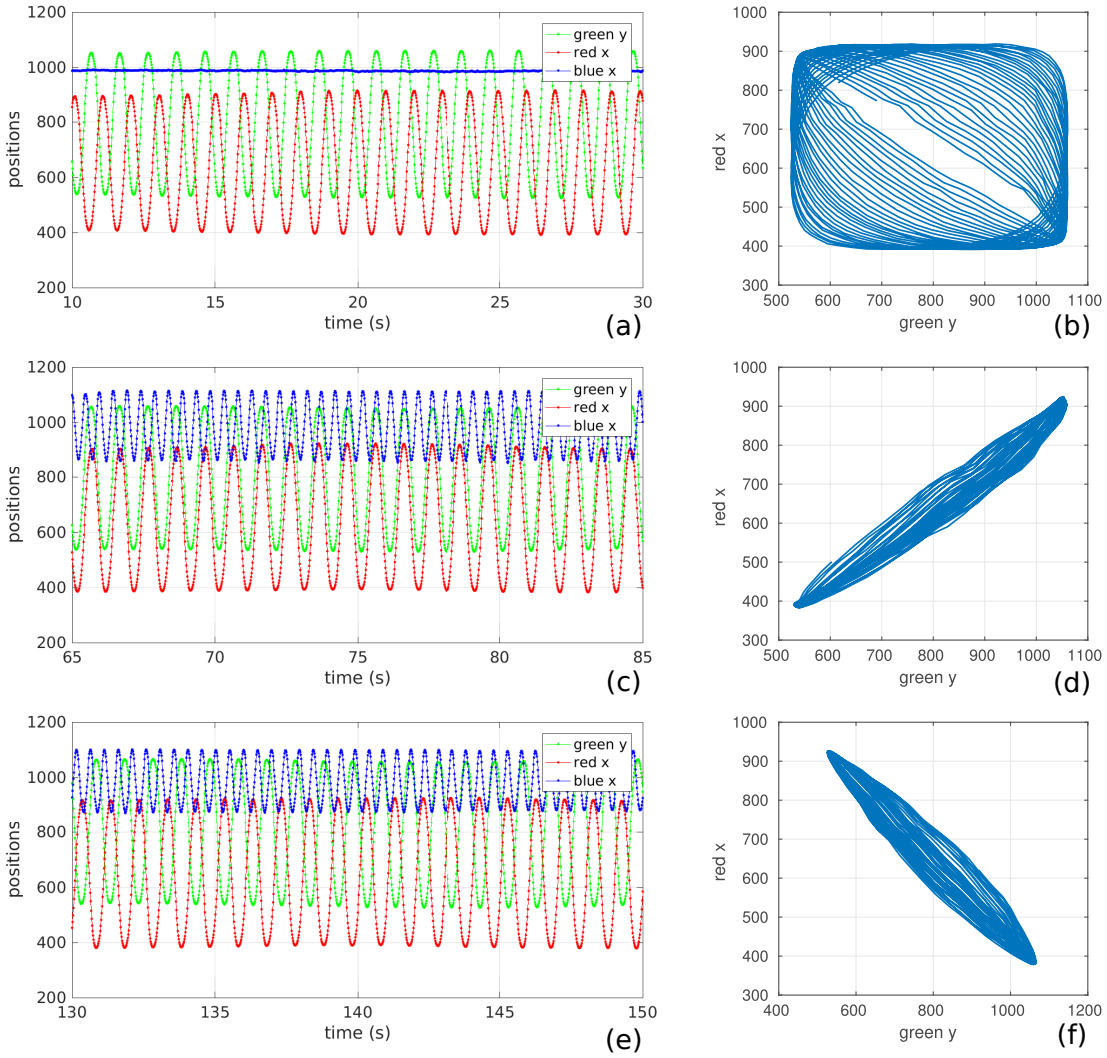


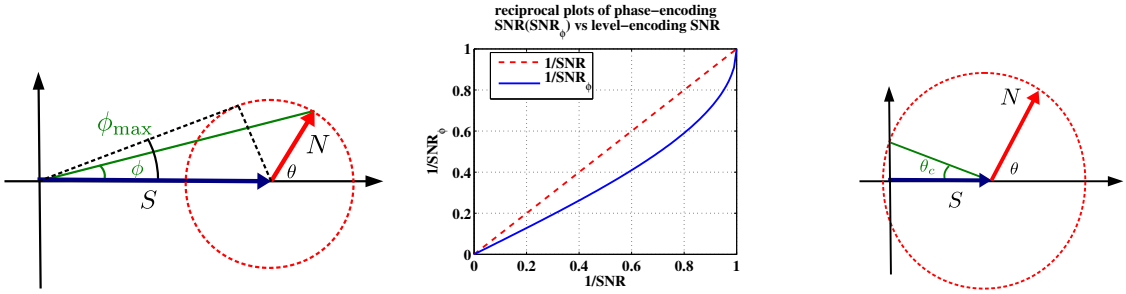

**Fig. 3.49:** Excerpts from Fig. 3.48 and their corresponding Lissajous curves.

From above, in this section we have shown the generality of the key mechanism of oscillator-based Boolean computation by demonstrating bit storage in mechanical metronomes. Through a creative setup, two sub-harmonically injection-locked metronomes have been shown to develop a bistable phase difference of either 0 or 180°, achieving a phase-encoded one-bit mechanical memory. As injection locking is a ubiquitous phenomenon in all types of oscillators, this demonstration has the potential of inspiring more implementations of phase-encoded logic latches using oscillator technologies from various physical domains.

## 3.4 Discussion on Oscillator-based Boolean Computation

### 3.4.1 Inherent Noise Immunity of Phase-based Logic

One of the key attractions of encoding logic in the phase of oscillatory signals is that, compared to level-based schemes, phase encoding provides inherent resistance to errors caused by additive noise and interference. There are two aspects to phase encoding that provide intrinsic resistance to additive noise/interference: 1) the effective SNR[10] for phase is increased by a factor of $\frac{\pi}{2}$ over SNR for level-based encodings, and 2) the oscillatory nature of the signal whose phase encodes the information makes much of the additive noise/interference average out, leading to smaller bit error rates than for the level-based case. These mechanisms are explained below.




(a) 1st mechanism: vector addition increases noise immunity in the phase domain.

(b) 1st mechanism: $\frac{1}{\text{“phase SNR”}}$ (blue) vs. $\frac{1}{\text{(level-based) SNR}}$ (red).

(c) 2nd immunity mechanism: averaging over $\theta$ reduces bit error rate.

**Fig. 3.50:** Mechanisms enhancing the noise resistance of phase-encoded logic.

Fig. 3.50(a) depicts an oscillatory signal as a phasor [98] $\vec{S}$, superimposed upon which is a noise (or interference) component, $\vec{N}$. The impact of this noise on the phase of the signal is shown by the phase error $\phi = \angle(\vec{S}+\vec{N})$. Given fixed amplitudes[11] $S = |\vec{S}|$ and $N = |\vec{N}|$, $\phi$

[10] Signal to noise ratio.

[11] For illustrative simplicity, we consider noise of only a fixed magnitude. In reality, of course, the magnitude of $\vec{N}$ has a probability distribution, *e.g.*, a Gaussian one.

depends on the relative angle $\theta$ between $\vec{S}$ and $\vec{N}$, *i.e.*, $\phi = g(\theta, \frac{N}{S})$[12]. For $N < S$ and fixed $N/S$, there is a maximum phase error over all $\theta$, *i.e.*, $\phi_{\max} = \sin^{-1}(N/S)$, as depicted in Fig. 3.50(a). The "phase SNR" is given by $\text{SNR}_\phi = \frac{\frac{\pi}{2}}{\phi_{\max}}$; it is the fraction, in angular terms, of the first quadrant taken up by the maximum phase error. This is to be compared against $S/N$, the SNR for level-based logic (for which, in Fig. 3.50(a), $\vec{N}$ is collinear with $\vec{S}$). As can be seen from Fig. 3.50(b), $\frac{1}{\text{SNR}_\phi}$ is always smaller than $\frac{1}{\text{SNR}}$, *i.e.*, the "phase SNR" is always improved over the standard level-based SNR. For small $S/N$ (*i.e.*, a large level-based SNR), this improvement is a factor of $\frac{\pi}{2} \simeq 1.6$. This is the first mechanism by which phase encoding improves noise immunity.

A second mechanism, conferring additional noise immunity, stems from that the phasors $\vec{S}$ and $\vec{N}$ are not necessarily always at the same frequency (as assumed implicitly in the analysis of the first mechanism, above), but can rotate at different speeds[13]. Consider now the case where the rotation speeds are very different. The rapid relative change in the angle $\theta$ between $\vec{S}$ and $\vec{N}$ suggests that the worst-case phase error $\phi_{\max}$, from the first mechanism above, is unduly pessimistic; and that, instead, the phase error *averaged over all values of* $\theta$ is the appropriate measure. More precisely, the standard deviation $\sigma_\phi$ that results from, *e.g.*, uniformly distributed $\theta$, is an appropriate measure of the phase error. This quantity can be substantially smaller than the worst-case phase error $\phi_{\max}$, implying considerable additional immunity to noise.

Indeed, this second mechanism makes phase encoding useful even when the noise magnitude is *greater* than that of the signal, a situation where level-based logic encoding becomes largely useless. This situation is illustrated in Fig. 3.50(c). Observe that for most values of $\theta$, the phase error is less than $\frac{\pi}{2}$, the threshold for a bit error. The probability of logical error in the case of phase encoding is $\frac{\theta_c}{\pi} = \frac{\cos^{-1}(S/N)}{\pi}$, which can be very small if $N$ is only slightly greater than $S$ (as depicted); and reaches its maximum, 50%, only as the noise increases to infinity. In contrast, the probability of logical error for level-based encoding is always 50% when $N > S$, since a logical error *always* results when the noise subtracts from the signal (rather than adding to it).

These noise immunity features of phase encoding do not come as a surprise; the superior noise properties of phase and frequency modulation (PM and FM), over those of amplitude modulation (AM), have long been known [100] and exploited in practice, *e.g.*, in radio communications. However, the authors are not aware of their prior realization, or application, in the context of logic encoding for general-purpose computing.

[12]For example, $g(0,\cdot) = g(\pi.\cdot) = 0$ and $g(\frac{pi}{2}, \frac{N}{S}) = \tan^{-1}(\frac{N}{S})$.

[13]The phasor $\vec{N}$ can be thought of as one component, at frequency $f$, of a spectral expansion [99] of a stochastic process.

### 3.4.2 Potential Power/energy Advantages of PHLOGON

PHLOGON offers significant energy-efficiency benefits over von Neumann's original scheme. It uses continuously-running oscillators, which can be much more energy-efficient than von Neumann's latch-rings. Moreover, neither distribution nor modulation of AC power is involved for running a PHLOGON architecture[14]. This reduces parasitic-related losses especially for large, intricately-routed systems, resulting in significant power savings over von Neumann's scheme.

Compared with level-based CMOS computation architecture, the circuits and nanodevice embodiments of PHLOGON can potentially still be considerably more energy efficient. Dynamic (capacitive charging/discharging) and continuous (sub-threshold leakage) power consumption in level-based CMOS are both strongly determined by the supply voltage. The lowest practical supply voltage today for level-based CMOS is about 0.8V; this number is unlikely to drop significantly in future years, due to threshold voltage, variability and noise barriers [101, 102]. In contrast, ring oscillators in standard CMOS technologies operate in sub-threshold mode at supply voltages as low as 80mV [103–105]; while in III-V technologies, ring oscillators running at 0.23V were demonstrated almost 30 years ago [106]. $10\times$ lower supply voltage translates to $100\times$ lower dynamic ($CV^2$) power, and more than $20,000\times$ lower leakage power (exponential in supply voltage). We emphasize that these power savings result simply by moving from level-based to phase-based logic architectures, without any change in the underlying CMOS technology.

Such large power savings can result even with ring oscillators, which dissipate most or all of their energy every cycle. When harmonic oscillators, with $Q$ factors appreciably greater than 1, are used, further energy savings[15] can be realized. On-chip CMOS LC oscillators with spiral inductors, though considerably larger in area than ring oscillators, are available today with $Q$ factors greater than 10, making them an interesting candidate to explore for additional power efficiency. Integrated MEMS resonators, though even larger in area, feature $Q$ factors of $10^4$-$10^5$ [89], potentially making them extremely attractive for low power computation with easily available and well-developed conventional technologies. Resonant Body Transistor (RBT), a silicon-based resonator compatible with standard CMOS process, has been demonstrated to achieve $>$10GHz frequency with $Q$ factor of 1830 [107], making it another promising candidate. Spin-torque nano-oscillators (STNOs) feature $Q$ factors of more than $10^4$ at frequencies of 25GHz [108–110]; as such, they offer very exciting power, as well as speed, possibilities.

[14] Note that SYNC and CLK can be weak, dissipating negligible power.

[15] *i.e.*, an energy advantage of roughly $Q$ over ring oscillators using the same technology and supply voltage.

### 3.4.3 Power-speed Trade-offs in PHLOGON Systems

#### 3.4.3.1 Power-speed Trade-offs in Oscillator Latches

In oscillator-based computing, the speed in which on oscillator arrives at the injection-locked state directly determines how fast bits can flip, a key property in computing. In Sec. 3.3.1, when demonstrating the operation of phase-based state machine using CMOS rings oscillators, we noted that for ring oscillators, the phase of the oscillator can be flipped within one cycle of oscillation. For better energy efficiency, we are also interested in using LC oscillators for PHLOGON. Therefore, the speed in which LC oscillators injection lock requires more in-depth investigation; it is the key question to answer for studying the speed of oscillator latches.

Intuitively, high-$Q$ LC oscillators are slower in response. Which is to say that its amplitude is often more stable and settles more slowly to its steady state. But is it also true for its phase? With a periodic injection, will a high-$Q$ oscillator's phase shift more slowly to its injection-locked state than one with a lower-$Q$?

To answer this question, we first have to define the Q factor of oscillators. Then we simulate a simple negative-resistance LC oscillator with an adjustable $Q$ factor and analyze the results.

**Rigorous $Q$ factor definition**

When we refer to an oscillator as having a high $Q$ factor, what we are often trying to say is that it has a stable frequency and amplitude. These properties often translate to better energy efficiency and lower phase noise, so the "quality" $Q$ is higher. However, once we try to write down an exact formula for the $Q$ factor of an oscillator, several confusions arise.

Firstly, $Q$ factor is often defined under the context of (usually second-order) linear resonators, which are systems with damped oscillatory behaviors. There are several definitions. One is the frequency-to-bandwidth ratio of the resonator:

$$Q \stackrel{\text{def}}{=} \frac{f_r}{\Delta f} = \frac{\omega_r}{\Delta \omega}. \tag{3.7}$$

The formula implies that there is a Bode plot of the system with a resonance frequency, thus is only meaningful for stable linear systems with well-defined inputs and outputs. It is not directly applicable to oscillators which are by-definition autonomous and usually nonlinear. Another definition for $Q$ factor is from the energy perspective:

$$Q \stackrel{\text{def}}{=} 2\pi \times \frac{\text{Energy Stored}}{\text{Energy Dissipated per Cycle}}. \tag{3.8}$$

This assumes that there is damping in the oscillation, which is not true for self-sustaining oscillators. There are other definitions that directly map $Q$ to a parameter in the transfer function, but they are limited to linear resonators as well.

Another common confusion is that people often simply assume an oscillator to have the same $Q$ factor as the resonator it is using inside. For example, an LC-type oscillator is often said to have the same $Q$ as the RLC circuit in it. However, this is not true either. An obvious conterexample is that the use of a high-$Q$ resonator in a nonlinear oscillator doesn't always result in a high-$Q$ oscillator.

Therefore, in this section, we first have to define the $Q$ factor of an oscillator. We define it also from the energy perspective. Consider perturbing an amplitude-stable oscillator with a small amount of extra energy. The oscillator will settle back to its amplitude-stable oscillatory state, dissipating (or restoring) some small amount of energy every cycle. Then we define the ratio between the extra energy applied and the energy dissipated (or restored) every cycle as the $Q$ factor of the oscillator:

$$Q \stackrel{\text{def}}{=} 2\pi \times \frac{\text{Extra Energy Applied}}{\text{Extra Energy Dissipated per Cycle}}. \tag{3.9}$$

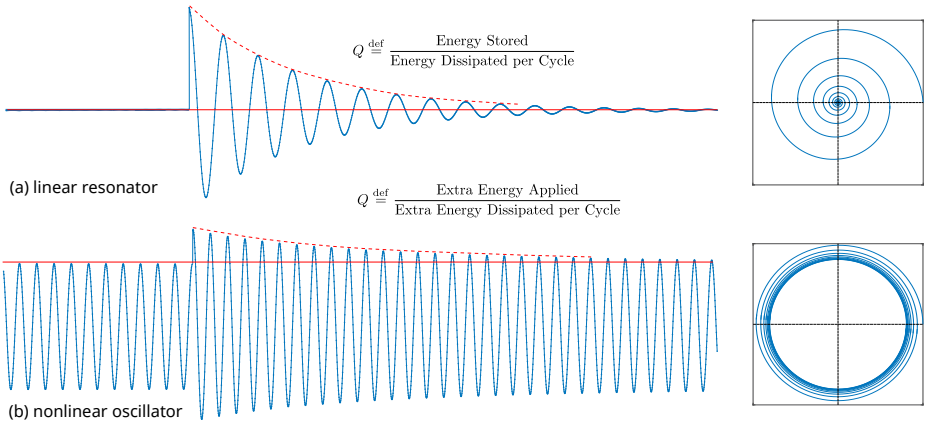



**Fig. 3.51:** Illustration of the definition of $Q$ factor for nonlinear oscillators.

The definition is illustrated in Fig. 3.51 with both the time-domain waveforms and phase plane plots. $Q$ can be conveniently measured from either transient simulations or experiment measurements, without analyzing circuit topology or open-loop block diagrams. The higher the $Q$ factor, the more slowly its amplitude responds to perturbations, and the more stable the limit cycle. These observations fit intuition.

As also illustrated in Fig. 3.51, the $Q$ definition for oscillators is analogous to that of the linear resonator. In the case of linear systems, the stable state is the zero state. And the energy-based $Q$ formulation as in (3.8) is indeed just a special case of our definition in (3.9), with the amplitude-stable state being the zero state. It is provable that for linear systems, the frequency-domain $Q$ definition in (3.7) is equivalent to (3.8) [16]. Therefore, our $Q$ formulation can be considered as a direct generalization from the two well-defined $Q$ factor formula for linear resonators.

[16]The two definitions in (3.7) and (3.8) have a fixed ratio for linear systems.

Note that because the oscillator is nonlinear, when we are measuring the $Q$ in the way shown in Fig. 3.51, the size of the extra amplitude introduced will affect the measurement. But as the extra amplitude gets smaller and smaller, the $Q$ factor defined in (3.9) should converge. We can then define the $Q$ factor more rigorously using this limit. The limit can be analyzed using techniques developed for Linear Period Time Varying (LPTV) systems. It can be estimated both analytically and numerically given the oscillator's equations [111]. Therefore, the $Q$ factor in (3.9) not only fits intuition, it is also a quantity that can be conveniently characterized and analyzed.

**Does It Take Longer to Injection Lock a High-$Q$ Oscillator?**

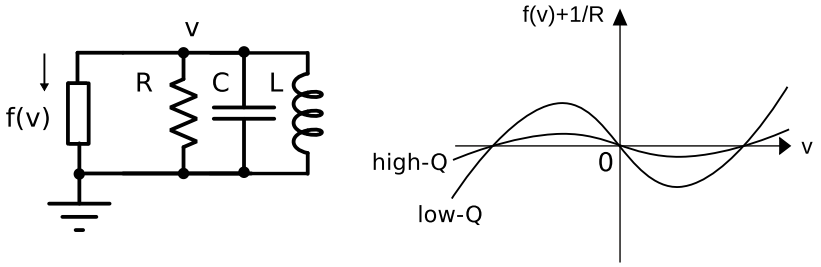



**Fig. 3.52:** A simple negative-resistance LC oscillator. Different choice of f(v) will result in different $Q$ factor of the oscillator.

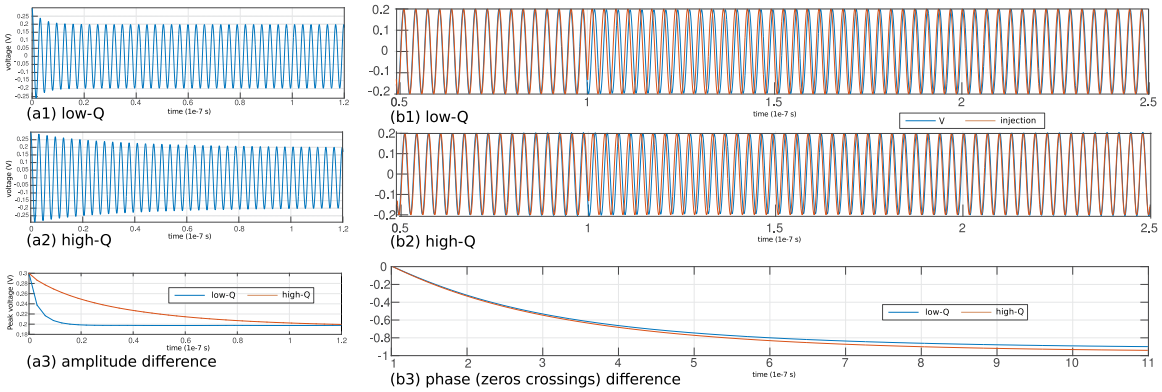



**Fig. 3.53:** Simulation results of amplitude (a1-3) and phase shifting (b1-3) of high-$Q$ and low-$Q$ oscillators. In (b3), we plot the zero-crossing differences between the v signals and injection signals in (b1) and (b2).

By definition, a high-$Q$ oscillator settles more slowly in amplitude. But is it also true for its phase? To rephrase the question: as $Q$ factor becomes higher, does it also take longer to injection lock the oscillator's phase?

To study this question, we consider a simple negative-resistance LC oscillator shown in Fig. 3.52 [112]. Different nonlinearities in $f(v)$ result in different $Q$ factors. Intuitively, as

$f(v)+\frac{1}{R}$ gets "flatter", the oscillator will appear more like an LC tank with no resistance, thus the $Q$ gets higher.

We simulate the LC oscillator with $L = 0.5nH$, $C = 0.5nF$, $f(v) = K \cdot (v - \tanh(1.01 \cdot v))$. We choose $K$ values as 1 and 20, the former results in a high-$Q$ oscillator, the latter low-$Q$. The simulation results in Fig. 3.53 show that the high-$Q$ one settles much more slowly in amplitude. Then we apply a small injection current $I(t) = 1mA \cdot \cos(\omega_0 t + \pi/2 * u(t - 100ns))$ at the only non-ground node of the circuit, where $\omega_0 = \frac{1}{\sqrt{LC}}$, $u(t)$ is the step function. In this way the injection signal shifts its phase to 90° after $100ns$. The oscillator's phase will follow this change by shifting gradually through the mechanism of injection locking. The results in Fig. 3.53 indicate that the difference in the phase shifting behavior between the high-$Q$ and low-$Q$ oscillators is marginal.

Therefore, at least in this preliminary experiment using LC oscillators, the shifting speeds of amplitude and phase seem to be decoupled. This result is intriguing as the speed in the phase shift doesn't seem to be sacrificed as we use more energy-efficient oscillators. This property is very appealing in the design of PHLOGON systems.

#### 3.4.3.2 Power-speed trade-off for Logic Gates

Fig. 3.54 depicts an example of a phase-based MAJORITY gate. We study the speed-power trade-off by setting up a series of simulations where we push the limit of this circuit in low-power or high-speed operations. The computational study here is done using the EKV MOSFET model in Spectre with EPFL 90nm model card. EKV model is well known for its accuracy in low-power applications.

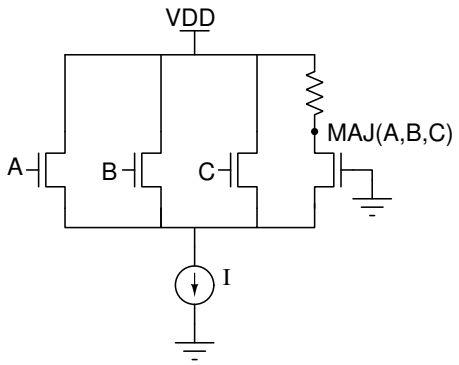



**Fig. 3.54:** A 3-input averager used as MAJORITY gate in phase-encoded logic computation.

With a fixed Vdd, as we reduce Is from the current source in the averager, power consumption decreases. In the meanwhile, we need to increase the load resistance to maintain output offset and gain. We keep gain fixed at its minimum at 1 to leverage the maximum bandwidth of the devices. Simulations show a clear speed-power trade-off, where the 3dB bandwidth is approximately proportional to Is. With transistors of 100nm length and 100nm width, 40nA Is results in 4GHz bandwidth; 10nA has 1GHz; 1nA has 0.1GHz; 10pA has 1MegHz.

Although in simulation we can reduce Is almost arbitrarily to cut energy consumption, the bandwidth of this circuit will quickly become too narrow for it to be useful as a logic gate.

Because of this speed-power trade-off, reducing power consumption requires either decreasing device sizes or reducing the supply voltage. Both options will worsen the signal-to-noise ratio, which brings about the trade-off with noise. As phase encoding has intrinsic immunity to noise, it is possible that the speed-power trade-off is better than that in conventional digital circuits. Combining the analysis on noise, energy, speed, and investigating PHLOGON systems in the full design space is still a direction for future research.

# Chapter 4

# Oscillator-based Ising Machines

In this chapter, we show that networks of coupled self-sustaining nonlinear oscillators can physically implement Ising machines, which are analog hardware suitable for solving many hard combinatorial optimization problems. Our scheme is soundly rooted in a novel theoretical result based on the phase-based analyses in Chapter 2 that connects the phase dynamics of coupled oscillator systems with the Ising Hamiltonian. Compared with existing Ising machine implementation proposals relying on futuristic (often quantum) device technologies, our scheme is amenable to realization using many kinds of oscillators from different physical domains, and is particularly well suited for CMOS, in which it offers significant practical advantages in scalability and miniaturizability.

The Ising model [113, 114] takes any weighted graph and uses it to define a scalar function called the Ising Hamiltonian. Each vertex in the graph is associated with a *spin*, *i.e.*, a binary variable taking values $\pm 1$. The Ising problem is to find an assignment of spins that minimises the Ising Hamiltonian (which depends on the spins and on the graph's weights). Solving the Ising problem in general has been shown to be very difficult [115], but devices that can solve it quickly using specialized hardware have been proposed in recent years [1, 116–120]. Such Ising machines have attracted much interest because many classically difficult combinatorial optimization problems (including all 21 of Karp's well-known list of NP-complete problems [121]) can be mapped to Ising problems [122]. Hence, as Moore's Law nears its limits, a physical implementation of coupled spins that can directly perform the minimization, namely an Ising machine, therefore becomes very attractive for potential speed and power advantages over conventional algorithms run on CPUs.

There are three types of existing Ising machine approaches. One is the D-WAVE quantum Ising machine [118, 119]. It uses superconducting loops as spins and requires a temperature below 80mK ($-273.07^{\circ}$C) to operate [118]. As a result, it needs a large footprint to accommodate the necessary cooling system. A second category of Ising machine proposals use novel non-quantum devices as Ising spins to achieve room-temperature operation. Most notable among them is a scheme known as CIM (Coherent Ising Machine), which uses laser pulses traveling on a multiple-kilometer-long optical fiber to represent Ising spins

[116, 117]. The coupling between these pulses is implemented digitally by measurement and feedback using multiple FPGAs. While CIM can potentially be more compact than D-WAVE's machines, it is unclear how it can be miniaturized and integrated due to the use of long fibers. Other researchers propose the use of several other novel devices as Ising spins, including MEMS resonators [123] and nanomagnets from Spintronics [124]. Physical realization of these machines still awaits future development of these emerging device technologies.

The third type of approaches build Ising model emulators using digital circuits [120, 125]. They are digital hardware implementations of the simulated annealing algorithm, and are thus not directly comparable to the other Ising machine implementations discussed above, which all attempt to use interesting intrinsic physics to minimize the Ising Hamiltonian for achieving large speedups.

Our work on oscillator-based Ising machine (OIM) [29, 126, 127] shows that almost all types of nonlinear self-sustaining oscillators are suitable to implement Ising machines physically. As many tried-and-tested types of such oscillators already exist, this scheme offers the advantages of scalability to large numbers of spins, high-speed and low-power operation, and straightforward design and fabrication using standard CMOS technology, unlike CIM and D-WAVE, which are large, expensive and ill-suited to low-cost mass production. Even though these advantages of CMOS also apply, of course, to hardware simulated annealing engines [120, 125], our scheme has additional attractive features. One key advantage relates to variability, a significant problem in nanoscale CMOS. For oscillator networks, variability impacts the system by causing a spread in the natural frequencies of the oscillators. Unlike other schemes, where performance deteriorates due to variability [120], we can essentially eliminate variability by means of simple VCO-based calibration to bring all the oscillators to the same frequency. Moreover, as we show in this chapter, our scheme is inherently resistant to variability even without such calibration. Another key potential advantage stems from the continuous/analog nature of our scheme (as opposed to purely digital simulated annealing schemes). Computational experiments indicate that the time our scheme takes to find good solutions of the Ising problem grows only very slowly with respect to the number of spins, offering significant potential advantages over simulated annealing schemes [125] as hardware sizes scale up to large numbers of spins.

In the remainder of this chapter, we first provide a brief summary of the Ising problem and existing Ising machine schemes in Sec. 4.1. We then present our oscillator-based Ising machine scheme (dubbed OIM, for Oscillator Ising Machine) in Sec. 4.2, explaining the theory that enables it to work. Then in Sec. 4.3, we present both computational and hardware examples showing the effectiveness of our scheme for solving several combinatorial optimization problems.

## 4.1 The Ising problem and Existing Ising Machine Approaches

The Ising model is named after the German physicist Ernest Ising. It was first studied in the 1920s as a mathematical model for explaining domain formation in ferromagnets [113]. It comprises a group of discrete variables $\{s_i\}$, *aka* spins, each taking a binary value $\pm 1$, such that an associated "energy function", known as the Ising Hamiltonian, is minimized:

$$\min \; H \triangleq -\sum_{1\le i<j\le n} J_{ij}s_is_j - \sum_{i=1}^{n} h_is_i, \quad \text{such that } \; s_i \in \{-1,\; +1\}, \tag{4.1}$$

where $n$ is the number of spins; $\{J_{ij}\}$ and $\{h_i\}$[1] are real coefficients.

The Ising model is often simplified by dropping the $\{h_i\}$ terms. Under this simplification, the Ising Hamiltonian becomes

$$H = -\sum_{i,j,\; i<j} J_{ij}s_is_j. \tag{4.2}$$

What makes the Ising model particularly interesting is that many hard optimization problems can be shown to be equivalent to it [128]. In fact, all of Karp's 21 NP-complete problems can be mapped to it by assigning appropriate values to the coefficients [122]. Physical systems that can directly minimize the Ising Hamiltonian, namely Ising machines, thus become very attractive for outperforming conventional algorithms run on CPUs for these problems.

Several schemes have been proposed recently for realizing Ising machines in hardware:

- Quantum annealers. Best-known examples of this kind are built by D-Wave Systems [118, 119]. Their quantum Ising machines use superconducting loops as spins and connect them using Josephson junction devices [129]. Their state of the art is a Ising machine with 2000 spins. As the machines require a temperature below 80mK ($-273.07^{\circ}$C) to operate [118], they all have a large footprint to accommodate the necessary cooling system. While many question their advantages over simulated annealing run on classical computers [130], proponents believe that through a mechanism known as quantum tunnelling, they can offer large speedups on problems with certain energy landscapes [131].

- Optical Coherent Ising Machines (CIM)

  Another approach to Ising machine implementation uses optical laser pulses as spins. When a photonic crystal[2] is pumped by a laser source, it can develop parametric oscillation at a subharmonic of the laser frequency. The resulting subharmonic pulse is known as an optical parametric oscillator (OPO). A sequence of such pulsed OPOs generated by the same laser source will each randomly assume one of two possible phase values separated

[1] $\{h_i\}$ coefficients are also known as self terms in the Ising Hamiltonian.

[2] Periodically Poled Lithium Niobate (PPLN) nonlinear crystals are used in such systems.

by 180°, constituting a stream of binary random numbers. Using mirrors and delays, these OPOs can be coupled to each other, as is shown in Fig. 4.1. In this coupled system, OPOs with certain phase configurations require a lower laser pumping power to excite — they represent the ground states in the Ising machine. To find them, the laser power can be slowly increased, and the first sequence of OPOs observed is expected to encode good solutions.[3]

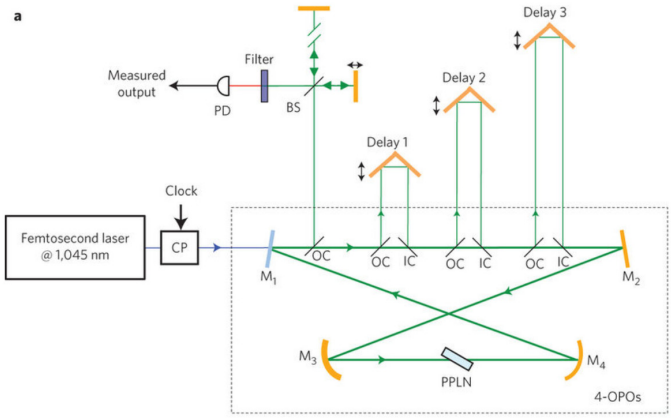



**Fig. 4.1:** Schematic for size-4 OPO-based Ising machine (Fig. 2 in [1]).

Implementations of OPO-based Ising machines have been growing in size over the recent years, from 4 spins [1], to 16 [132], 100 [116] and 2000 spins [117]. These systems require complex manual fabrication and assembly processes, especially compared to CMOS implementations. For example, miniaturization is difficult on account of the need for kilometer-long optical fibres.

- Other non-quantum Ising machine proposals. Recent studies have also proposed the use of several novel nanodevices as Ising spins, such as MEMS (Micro-Electro-Mechanical Systems) resonators [123]. Stochastic nanomagnets with low energy barriers are another candidate for implementing Ising spins [133]. In recent computational studies, they have been given the name "p-bits", and have been shown to minimize Ising Hamiltonians [124]. Such nanomagnets are conventionally used in magnetic random access memories (MRAMs). Hence, the proposed nanomagnet scheme resembles the aforementioned CMOS Ising chip using SRAMs. Since nanomagnet Ising machines do not appear to actually have been built (published results are based on simulations/emulations), it is unclear if the scheme suffers from variability issues that afflict CMOS Ising machines. Physical realization of these machines still awaits future development of these emerging device technologies.

- Ising machines implemented as CMOS digital circuits

[3] Alternatively, the Ising machine can also be operated with a fixed laser power while the coupling strength is ramped up.

Another broad direction is to build Ising model emulators using digital circuits. A recent implementation [120] uses CMOS SRAM cells as spins, and couples them using digital logic gates. The authors point out, however, that "the efficacy in achieving a global energy minimum is limited" [120] due to variability. The speed-up and accuracy reported by [120] are instead based on deterministic on-chip computation paired with an external random number generator — a digital hardware implementation of the simulated annealing algorithm. More recently, similar digital accelerators have also been tried on FPGAs [134]. These implementations are not directly comparable to the other Ising machine implementations discussed above, which attempt to use interesting intrinsic physics to minimize the Ising Hamiltonian for achieving large speedups.

## 4.2 Mechanism of Oscillator-based Ising Machines

In this section, we show that a network of coupled self-sustaining oscillators can function as an Ising machine and physically minimize the Ising Hamiltonian. The essence of the scheme is to interconnect oscillators with bistable phases induced by SHIL. A network of such binarized oscillators prefer certain binary configurations over others; we show that they correspond to the local minima of the Ising Hamiltonian (Fig. 4.2). Two simple additional steps (*i.e.*, adding noise, and turning SHIL on and off) enable the network to find excellent solutions of Ising problems.

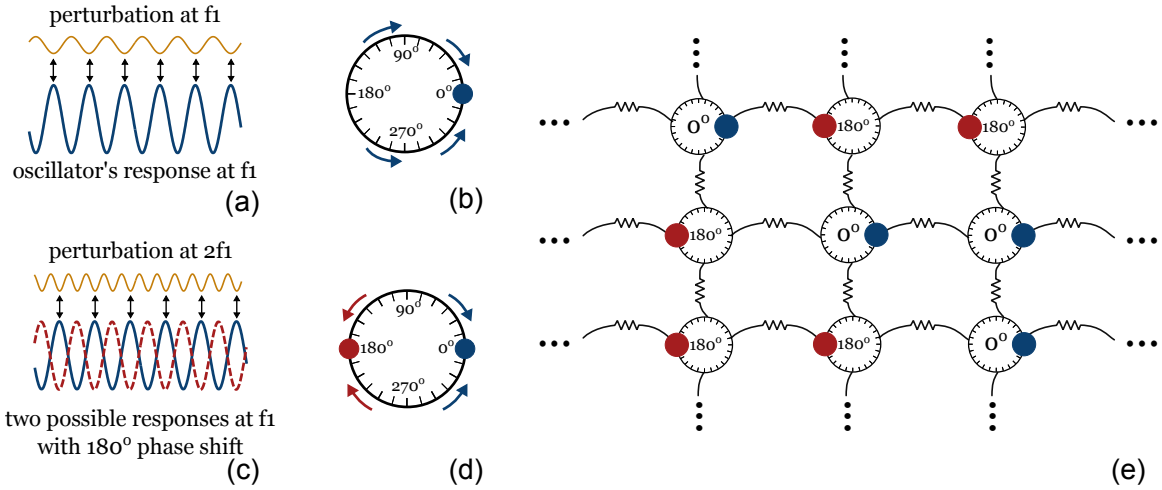



**Fig. 4.2:** Illustration of the basic mechanism of oscillator-based Ising machines: (a) oscillator shifts its natural frequency from $f_0$ to $f_1$ under external perturbation; (b) oscillator's phase becomes stably locked to the perturbation; (c) when the perturbation is at $2f_1$, the oscillator locks to its subharmonic at $f_1$; (d) bistable phase locks under subharmonic injection locking; (e) coupled subharmonically injection-locked oscillators settle with binary phases representing an optimal spin configuration for an Ising problem.

To understand the mechanism of such Ising machines, we need to answer the question: what does a coupled oscillator network minimize? In Sec. 2.3, we have already introduced the global Lyapunov function of coupled oscillator networks, which is a scalar "energy like" quantity that is naturally minimized by the system. Here we further show its connection to the Ising Hamiltonian in Sec. 4.2.1. Then in Sec. 4.2.2, we introduce SHIL into the system to binarize the phases of oscillators, and derive a new Lyapunov function that the system with SHIL minimizes. By examining this function's equivalence to the Ising Hamiltonian, we show that such a coupled oscillator network under SHIL indeed physically implements an Ising machine. In Sec. 4.2.3, we study the network's behavior in the presence of noise. Finally, in Sec. 4.2.4, we consider the effect of variability on the system's operation. We show that a spread in the natural frequencies of the oscillators contributes a linear term in the global Lyapunov function, and does affect Ising machine performance by much if the variability is not extreme.

### 4.2.1 Global Lyapunov Function of Coupled Oscillators

Recall the generalized Kuramoto model (2.38) for coupled oscillators

$$\frac{d}{dt}\phi_i(t) = \omega_i - \omega^* + \omega_i \cdot \sum_{j=1,\ j\neq i}^{n} c_{ij}(\phi_i(t) - \phi_j(t)). \tag{2.38}$$

Again, $\{\phi_i\}$ represents the phases of $n$ oscillators; $\omega_i$ is the frequency of the oscillator whereas $\omega^*$ is the central frequency of the network. $c_{ij}(.)$ is a $2\pi$-periodic function for the coupling between oscillators $i$ and $j$.

To simplify exposition, we now assume that the $c_{ij}$ functions are sinusoidal.[4] We further assume zero spread in the natural frequencies of oscillators, *i.e.*, $\omega_i \equiv \omega^*$, and discuss the effect of frequency variability later in Sec. 4.2.4. With these simplifications, (2.38) can be written as

$$\frac{d}{dt}\phi_i(t) = -K \cdot \sum_{j=1,\ j\neq i}^{n} J_{ij} \cdot \sin(\phi_i(t) - \phi_j(t)). \tag{4.3}$$

Here, we are using the coefficients $\{J_{ij}\}$[5] from the Ising model (4.1) to set the connectivity of the network, *i.e.*, the coupling strength between oscillators $i$ and $j$ is proportional to $J_{ij}$. The parameter $K$ modulates the overall coupling strength of the network.

There is a global Lyapunov function associated with (4.3) [135]:

$$E(\vec{\phi}(t)) = -K \cdot \sum_{i,j,\ i\neq j} J_{ij} \cdot \cos(\phi_i(t) - \phi_j(t)), \tag{4.4}$$

[4]This need not be the case for the analysis to hold true, as shown in Sec. 2.3. More generally, $c_{ij}$s can be any $2\pi$-periodic odd functions, which are better suited to practical oscillators.

[5]In the Ising Hamiltonian (4.1), $J_{ij}$ is only defined when $i < j$; here we assume that $J_{ij} = J_{ji}$ for all $i,\ j$.

where $\vec{\phi}(t) = [\phi_1(t), \cdots, \phi_n(t)]^T$. Being a global Lyapunov function, it is an objective function the coupled oscillator system always tends to minimize as it evolves over time [68].

If we look at the values of this continuous function $E(\vec{\phi}(t))$ at some discrete points, we notice that it shares some similarities with the Ising Hamiltonian. At points where every $\phi_i$ is equal to either 0 or $\pi$,[6] if we map $\phi_i = 0$ to $s_i = +1$ and $\phi_i = \pi$ to $s_i = -1$, we have

$$E(\vec{\phi}(t)) = -K \cdot \sum_{i,j,\ i\neq j} J_{ij} \cdot \cos(\phi_i(t) - \phi_j(t)) = -K \cdot \sum_{i,j,\ i\neq j} J_{ij} s_i s_j = -2K \cdot \sum_{i,j,\ i<j} J_{ij} s_i s_j. \tag{4.5}$$

If we choose $K = 1/2$, the global Lyapunov function in (4.4) exactly matches the Ising Hamiltonian in (4.2) at these discrete points. But this does not mean that coupled oscillators are naturally minimizing the Ising Hamiltonian, as there is no guarantee at all that the phases $\{\phi_i(t)\}$ are settling to these discrete points. In fact, networks with more than two oscillators almost always synchronize with analog phases, *i.e.*, $\{\phi_i(t)\}$ commonly settle to continuous values spread out in the phase domain as opposed to converging towards 0 and $\pi$. As an example, Fig. 4.3 (a) shows the phase responses of 20 oscillators connected in a random graph. As phases do not settle to the discrete points discussed above, the Lyapunov function they minimize becomes irrelevant to the Ising Hamiltonian, rendering the system ineffective for solving Ising problems. While one may think that the analog phases can still serve as solutions when rounded to the nearest discrete points, experiments in Sec. 4.3.2 show that the quality of these solutions is very poor compared with our scheme of Ising machines proposed here.

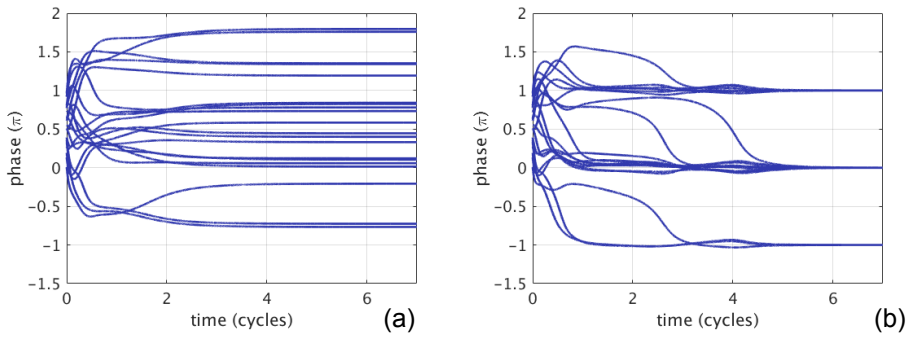



**Fig. 4.3:** Phases of 20 oscillators with random $\{J_{ij}\}$ generated by `rudy -rnd_graph 20 50 10001`: (a) without SYNC; (b) with $K_s = 1$.

### 4.2.2 Coupled Oscillators under SHIL and the Global Lyapunov Function

In our scheme, a common SYNC signal at $2\omega^*$ is injected to every oscillator in the network. Through the mechanism of SHIL, the oscillator phases are binarized. The example

[6] More generally, we can use $\{2k\pi \mid k \in \mathbf{Z}\}$ and $\{2k\pi + \pi \mid k \in \mathbf{Z}\}$ to represent the two states for each oscillator's phase.

shown in Fig. 4.3 (b) confirms that this is indeed the case: under SHIL, the phases of 20 oscillators connected in the same random graph now settle very close to discrete points. To write the model for such a system, we recall from Sec. 2.2 that a $2\omega^*$ perturbation introduces a $\pi$-periodic coupling term (*e.g.*, $\sin(2\phi)$) in the phase dynamics. Therefore, here we can directly write the model as follows.

$$\frac{d}{dt}\phi_i(t) = -K \cdot \sum_{j=1,\ j\neq i}^{n} J_{ij} \cdot \sin(\phi_i(t) - \phi_j(t)) - K_s \cdot \sin(2\phi_i(t)), \tag{4.6}$$

where $K_s$ represents the strength of coupling from SYNC.

Remarkably, there is a global Lyapunov function for this new type of coupled oscillator system. It can be written as

$$E(\vec{\phi}(t)) = -K \cdot \sum_{i,j,\ i\neq j} J_{ij} \cdot \cos(\phi_i(t) - \phi_j(t)) - K_s \cdot \sum_{i=1}^{n} \cos(2\phi_i(t)). \tag{4.7}$$

Now, we show that $E$ in (4.7) is indeed a global Lyapunov function. To do so, we again differentiate $E$ with respect to $\vec{\phi}$.

$$\begin{aligned}
\frac{\partial E(\vec{\phi}(t))}{\partial \phi_k(t)} =& -K \cdot \sum_{l=1,\ l\neq k}^{n} J_{kl} \frac{\partial}{\partial \phi_k(t)} [\cos(\phi_k(t) - \phi_l(t))] - K \cdot \sum_{l=1,\ l\neq k}^{n} J_{lk} \frac{\partial}{\partial \phi_k(t)} [\cos(\phi_l(t) - \phi_k(t))] \\
& - K_s \cdot \frac{\partial}{\partial \phi_k(t)} \cos(2\phi_k(t)) && (4.8)\\
=& K \cdot \sum_{l=1,\ l\neq k}^{n} J_{kl} \sin(\phi_k(t) - \phi_l(t)) - K \cdot \sum_{l=1,\ l\neq k}^{n} J_{lk} \sin(\phi_l(t) - \phi_k(t)) + K_s \cdot 2 \cdot \sin(2\phi_k(t)) && (4.9)\\
=& K \cdot \sum_{l=1,\ l\neq k}^{n} J_{kl} \cdot 2 \cdot \sin(\phi_k(t) - \phi_l(t)) + K_s \cdot 2 \cdot \sin(2\phi_k(t)) && (4.10)\\
=& -2 \cdot \frac{d\phi_k(t)}{dt}. && (4.11)
\end{aligned}$$

Therefore,

$$\begin{aligned}
\frac{\partial E(\vec{\phi}(t))}{\partial t} &= \sum_{k=1}^{n} \left[ \frac{\partial E(\vec{\phi}(t))}{\partial \phi_k(t)} \cdot \frac{d\phi_k(t)}{dt} \right] && (4.12)\\
&= -2 \cdot \sum_{k=1}^{n} \left( \frac{d\phi_k(t)}{dt} \right)^2 \leq 0. && (4.13)
\end{aligned}$$

Thus, we have proved that (4.7) is indeed a global Lyapunov function the coupled oscillators under SHIL naturally minimize over time.

At the discrete points (phase values of $0/\pi$), because $\cos(2\phi_i) \equiv 1$, (4.7) reduces to

$$E(\vec{\phi}(t)) \approx -K \cdot \sum_{i,j,\ i\neq j} J_{ij} \cdot \cos(\phi_i(t) - \phi_j(t)) - n \cdot K_s, \tag{4.14}$$

where $n \cdot K_s$ is a constant. By choosing $K = 1/2$, we can then make (4.14) equivalent to the Ising Hamiltonian in (4.2) with a constant offset.

Note that the introduction of SYNC does not change the relative $E$ levels between the discrete points, but modifies them by the *same* amount. However, with SYNC, all phases can be forced to eventually take values near either 0 or $\pi$ — the system now tries to reach a binary state that minimizes the Ising Hamiltonian, thus functioning as an Ising machine. We emphasize that this is *not* equivalent to running the system without SHIL and then rounding the analog phase solutions to discrete values as a post-processing step. Instead, the introduction of SHIL modifies the energy landscape of $E$, changes the dynamics of the coupled oscillator system, and as we show in Sec. 4.3, results in greatly improved minimization of the Ising Hamiltonian.

It is worth noting, also, that the Lyapunov function in (4.7) will, in general, have many local minima and there is no guarantee the oscillator-based Ising machine will settle at or near any global optimal state. However, as we show in Sec. 4.2.3, when judicious amounts of noise are introduced via a noise level parameter $K_n$, it becomes more likely to settle to lower minima. Indeed, the several parameters in the Ising machine — $K$, $K_s$ and $K_n$ — all play an important role in its operation and should be given suitable values. Furthermore, $K$, $K_s$, $K_n$ can also be time varying, creating various "annealing schedules". As we show in Sec. 4.3, this feature gives us considerable flexibility in operating oscillator-based Ising machines for good performance.

### 4.2.3 Stochastic Model of Oscillator-based Ising Machines

Noise in the phases of oscillators is commonly modelled by adding white noise sources to the oscillator frequencies:

$$\frac{d}{dt}\phi_i(t) = -K \cdot \sum_{j=1,\ j\neq i}^{n} J_{ij} \cdot \sin(\phi_i(t) - \phi_j(t)) - K_s \cdot \sin(2\phi_i(t)) + K_n \cdot \xi_i(t), \tag{4.15}$$

where variable $\xi_i(t)$ represents Gaussian white noise with a zero mean and a correlator $\langle \xi_i(t),\ \xi_i(\tau) \rangle = \delta(t-\tau)$; scaler $K_n$ represents the magnitude of noise.

(4.15) can be rewritten as a stochastic differential equation (SDE).

$$d\phi_{it} = \left[ -K \cdot \sum_{j=1,\ j\neq i}^{n} J_{ij} \cdot \sin(\phi_{it} - \phi_{jt}) - K_s \cdot \sin(2\phi_{it}) \right] dt + K_n \cdot dW_t, \tag{4.16}$$

and can then be simulated with standard SDE solvers.

To analyse the steady states of this SDE, we can apply the Boltzmann law from statistical mechanics [136]. For a system with discrete states $\vec{s}_i$, $i = 1, \cdots, M$, if each state is associated with an energy $E_i$,[7] the probability $P_i$ for the system to be at each state can be written as follows.

$$P_i = \frac{e^{-E_i/kT}}{\sum_{j=1}^{M} e^{-E_j/kT}}, \tag{4.17}$$

where $k$ is the Boltzmann constant, $T$ is the thermodynamic temperature of the system. While $k$ and $T$ are concepts specific to statistical mechanics, in this context the product $kT$ corresponds to the noise level $K_n$.

Given two spin configurations $\vec{s}_1$ and $\vec{s}_2$, the ratio between their probabilities is known as the Boltzmann factor:

$$\frac{P_2}{P_1} = e^{\frac{E_1 - E_2}{kT}}. \tag{4.18}$$

According to the energy function (4.14) associated with oscillator-based Ising machines, the energy difference that determines this factor is proportional to the coupling strength.

$$E_1 - E_2 \propto K. \tag{4.19}$$

If $\vec{s}_1$ is the higher energy state, *i.e.*, $E_1 > E_2$, as the coupling strength $K$ increases, it becomes less and less likely for the system to stay at $\vec{s}_1$. The system prefers the lowest energy state in the presence of noise.

### 4.2.4 Coupled Oscillator Networks with Frequency Variations

A major obstacle to the practical implementation of large-scale Ising machines is variability. While few analyses exist for assessing the effects of variability for previous Ising machine technologies, the effect of variability on our oscillator-based Ising machine scheme is easy to analyze, predicting that performance degrades gracefully.

One very attractive feature of oscillators is that variability, regardless of the nature and number of elemental physical sources, eventually manifests itself essentially in only one parameter, namely the oscillator's natural frequency. As a result, the effect of variability in an oscillator network is that there is a spread in the natural frequencies of the oscillators. Taking this into consideration, our model can be revised as

$$\frac{d}{dt}\phi_i(t) = \omega_i - \omega^* - \omega_i \cdot K \cdot \sum_{j=1,\ j\neq i}^{n} J_{ij} \cdot \sin(\phi_i(t) - \phi_j(t)) - \omega_i \cdot K_s \cdot \sin(2\phi_i(t)). \tag{4.20}$$

As it turns out, there is also a global Lyapunov function associated with this system.

$$E(\vec{\phi}(t)) = -K \cdot \sum_{i,j,\ i\neq j} J_{ij} \cdot \cos(\phi_i(t) - \phi_j(t)) - K_s \cdot \sum_{i=1}^{n} \cos(2\phi_i(t)) - 2\sum_{i=1}^{n} \frac{\omega_i - \omega^*}{\omega_i}\phi_i. \tag{4.21}$$

[7] It is provable that a global Lyapunov function, if it exists, can be used instead of a physical energy to derive the same Boltzmann law [137].

This can be proven as follows.

$$\frac{\partial E(\vec{\phi}(t))}{\partial \phi_k(t)} = K \cdot \sum_{l=1,\ l\neq k}^{n} J_{kl} \cdot 2 \cdot \sin(\phi_k(t) - \phi_l(t)) + K_s \cdot 2 \cdot \sin(2\phi_k(t)) - 2\frac{\omega_k - \omega^*}{\omega_k} \quad (4.22)$$

$$= -\frac{2}{\omega_k} \cdot \frac{d\phi_k(t)}{dt}. \quad (4.23)$$

Therefore,

$$\frac{dE(\vec{\phi}(t))}{dt} = -\sum_{k=1}^{n} \frac{2}{\omega_k} \left(\frac{d\phi_k(t)}{dt}\right)^2 \leq 0. \quad (4.24)$$

Note that (4.21) differs from (4.7) only by a weighted sum of $\phi_i$ — it represents essentially the same energy landscape but tilted linearly with the optimization variables. While it can still change the locations and values of the solutions, its effects are easy to analyse given a specific combinatorial optimization problem. Also, as the coupling coefficient $K$ gets larger, the effect of variability can be reduced. Small amounts of variability merely perturb the locations of minima a little, *i.e.*, the overall performance of the Ising machine remains essentially unaffected. Very large amounts of variability can, of course, eliminate minima that would exist if there were no variability. However, another great advantage of using oscillators is that even in the presence of large variability, the frequency oscillators can be calibrated (*e.g.*, using a voltage-controlled oscillator (VCO) scheme) prior to each run of the machine. As a result, the spread in frequencies can be essentially eliminated in a practical and easy-to-implement way.

## 4.3 Examples

In this section, we demonstrate the feasibility and efficacy of our oscillator-based Ising machine scheme through examples in both simulation and hardware.

### 4.3.1 Small MAX-CUT Problems

Given an undirected graph, the MAX-CUT problem [138, 139] asks us to find a subset of vertices such that the total weights of the cut set between this subset and the remaining vertices are maximized. As an example, Fig. 4.4 shows a size-8 cubic graph, where each vertex is connected to three others — neighbours on both sides and the opposing vertex. As shown in Fig. 4.4, dividing the 8 vertices randomly yields a cut size of 5; grouping even and odd vertices, which one may think is the best strategy, results in a cut size of 8; the maximum cut is actually 10, with one of the solutions shown in the illustration. Changing the edge weights to non-unit values can change the maximum cut and also make the solution look less regular, often making the problem more difficult to solve. While the problem may not seem challenging at size 8, it quickly becomes intractable as the size of the graph grows. In fact, MAX-CUT is one of Karp's 21 NP-complete problems [121].

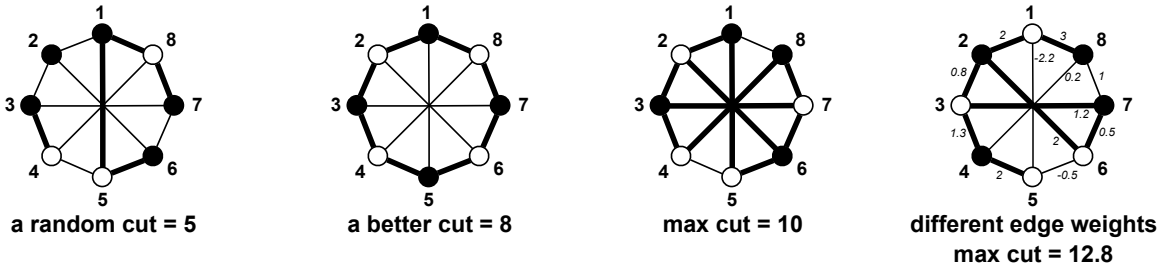



**Fig. 4.4:** Illustration of different cut sizes in a 8-vertex cubic graph with unit edge weights, and another one with random weights (rightmost).

The MAX-CUT problem has a direct mapping to the Ising model [121], by choosing $J_{ij}$ to be the opposite of the weight of the edge between vertices $i$ and $j$, *i.e.*, $J_{ij} = -w_{ij}$. To explain this mapping scheme, we can divide the vertices into two sets — $V_1$ and $V_2$. Accordingly, all the edges in the graph are separated into three groups — those that connect vertices within $V_1$, those within $V_2$, and the cut set containing edges across $V_1$ and $V_2$. The sums of the weights in these three sets are denoted by $S_1$, $S_2$ and $S_{cut}$. Together, they constitute the total edge weights of the graph, which is also the negation of the sum of all the $J_{ij}$s:

$$S_1 + S_2 + S_{cut} = \sum_{i,j,\ i<j} w_{ij} = -\sum_{i,j,\ i<j} J_{ij}. \tag{4.25}$$

We then map this division of vertices to the values of Ising spins, assigning $+1$ to a spin $i$ if vertex $v_i \in V_1$, and $-1$ if the vertex is in $V_2$. The Ising Hamiltonian in (4.2) can then be calculated as

$$\begin{aligned} H &= \sum_{i,j,\ i<j} J_{ij} s_i s_j \\ &= \sum_{i<j,\ v_i,v_j \in V_1} J_{ij}(+1)(+1) + \sum_{i<j,\ v_i,v_j \in V_2} J_{ij}(-1)(-1) + \sum_{i<j,\ v_i \in V_1, v_j \in V_2} J_{ij}(+1)(-1) \\ &= \sum_{i<j,\ v_i,v_j \in V_1} J_{ij} + \sum_{i<j,\ v_i,v_j \in V_2} J_{ij} - \sum_{i<j,\ v_i \in V_1, j \in V_2} J_{ij} \\ &= -\left(S_1 + S_2 - S_{cut}\right) = \sum_{i,j,\ i<j} J_{ij} - 2 \cdot S_{cut}. \end{aligned} \tag{4.26}$$

Therefore, when the Ising Hamiltonian is minimized, the cut size is maximized.

To show that an oscillator-based Ising machine can indeed be used to solve MAX-CUT problems, we simulated the Kuramoto model in (4.6) while making the $J_{ij}$s represent the unit-weight cubic graph in Fig. 4.4. The magnitude of SYNC is fixed at $K_s = 3$, while we ramp up the coupling strength $K$ from 0 to 5. Results from the deterministic model ($K_n = 0$) and the stochastic model ($K_n = 0.1$) are shown in Fig. 4.5 and Fig. 4.6 respectively. In the simulations, oscillators started with random phases between 0 and $\pi$;

after a while, they all settled to one of the two phase-locked states separated by $\pi$. These two groups of oscillators represent the two subsets of vertices in the solution. The results for the 8 spins shown in Fig. 4.5 and Fig. 4.6 are $\{+1,-1,+1,-1,-1,+1,-1,+1\}$ and $\{-1,+1,+1,-1,+1,-1,-1,+1\}$ respectively; both are global optimal solutions.

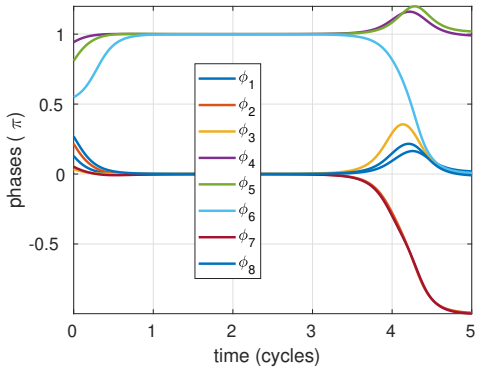



**Fig. 4.5:** Phases of oscillators solving a size-8 MAX-CUT problem without noise.

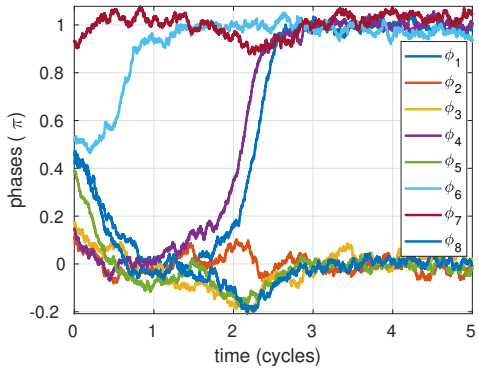



**Fig. 4.6:** Phases of oscillators solving a size-8 MAX-CUT problem with noise.

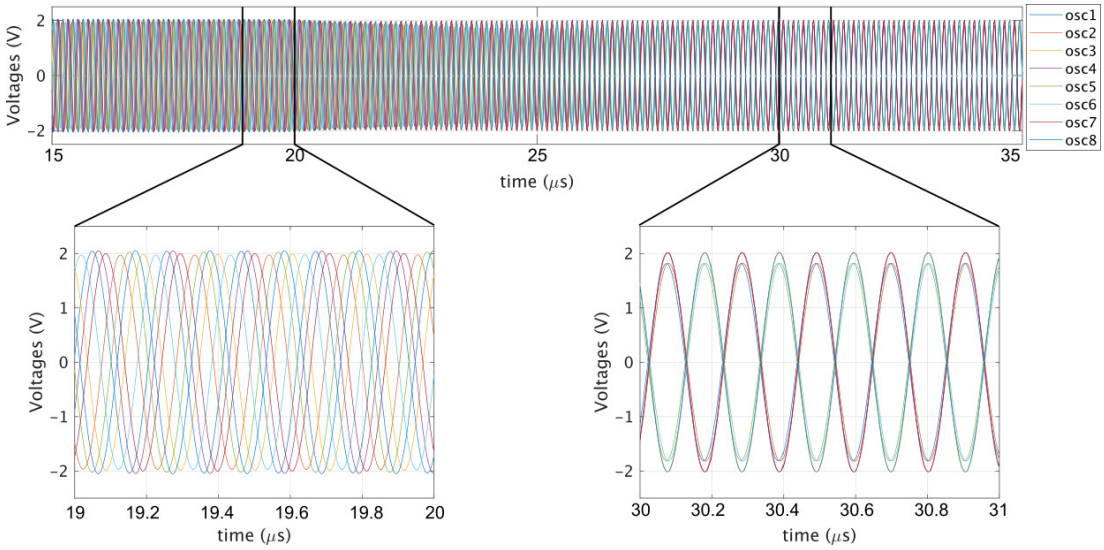



**Fig. 4.7:** Simulation results from ngspice on 8 coupled oscillators.

We have also directly simulated coupled oscillators at the SPICE level to confirm the results obtained on phase macromodels. Such simulations are at a lower lever than phase macromodels and are less efficient. But they are closer to physical reality and are useful for circuit design. In the simulations, 8 cross-coupled LC oscillators are tuned to a frequency of 5MHz. They are coupled through resistors, with conductances proportional to the coupling coefficients; in this case, we use $J_{ij} \cdot 1/100k\Omega$. Results from transient simulation using ngspice-28 are shown in Fig. 4.7. The 8 oscillators' phases settle into the two groups

{1,4,6,7} and {2,3,5,8}, representing one of the optimal solutions for the MAX-CUT problem. They synchronize within 20$\mu$s after oscillation starts, which is about 100 cycles. We have tried this computational experiment with different random initial conditions; like phase-macromodels, the SPICE-level simulations of these coupled oscillators reliably return optimal solutions for this size-8 MAX-CUT problem.

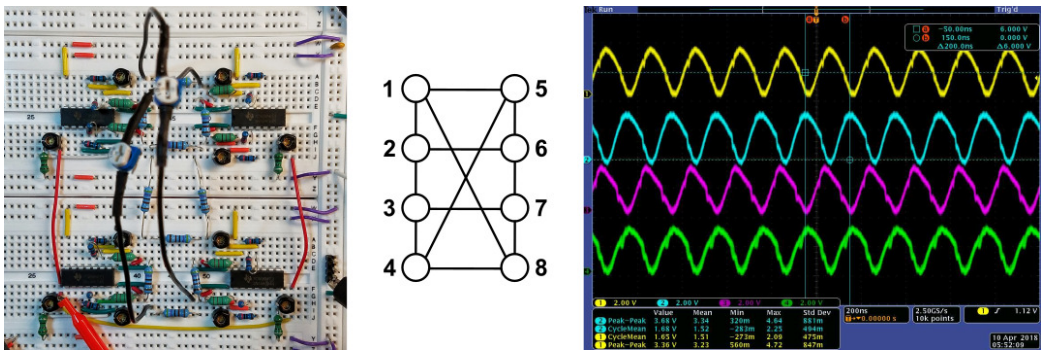



**Fig. 4.8:** A simple oscillator-based Ising machine solving size-8 cubic graph MAX-CUT problems: (a) breadboard implementation with 8 CMOS LC oscillators; (b) illustration of the connections; (c) oscilloscope measurements showing waveforms of oscillator 1∼4.

We have also implemented these 8 coupled LC oscillators on a breadboard; a photo of it is shown in Fig. 4.8. The inductance of the LC oscillators is provided by fixed inductors of size 33$\mu$H. The capacitance is provided by trimmer capacitors with a maximum value of 50pF; we have tuned them to around 30pF such that the natural frequencies of all oscillators are about 5MHz. The nonlinearity for sustaining the LC oscillation is implemented by cross-coupled CMOS inverters on TI SN74HC04N chips. SYNC is supplied through the GND pins of these chips. The results have been observed using two four-channel oscilloscopes; a screenshot of one of them is shown in Fig. 4.8. Through experiments with various sets of edge weights, we have validated that this is indeed a proof-of-concept hardware implementation of oscillator-based Ising machines for size-8 cubic-graph MAX-CUT problems.

Using the same type of oscillators, we have built hardware Ising machines of larger sizes. Fig. 4.9 shows a size-32 example implementing a type of connectivity known as the Chimera graph, much like the quantum Ising machines manufactured by D-Wave Systems. In this graph, oscillators are organized into groups of 8, with denser connections within the groups and sparse ones in between. The hardware is on perfboards, with components soldered on the boards so that the setup is more permanent than those on breadboards. Connections are implemented using rotary potentiometers. Next to each potentiometer we have designed male pin connectors soldered on the board such that the polarity of each connection can be controlled by shorting different pins using female jumper caps. When encoding Ising problems, we have also color-coded the jumper caps to make debugging easier, as can be seen in the photo as red and green dots next to the four arrays of white round potentiometers. To read the phases of the oscillators, instead of using multichannel oscilloscopes, we have soldered TI SN74HC86N Exclusive-OR (XOR) gate chips on board. The XOR operation of

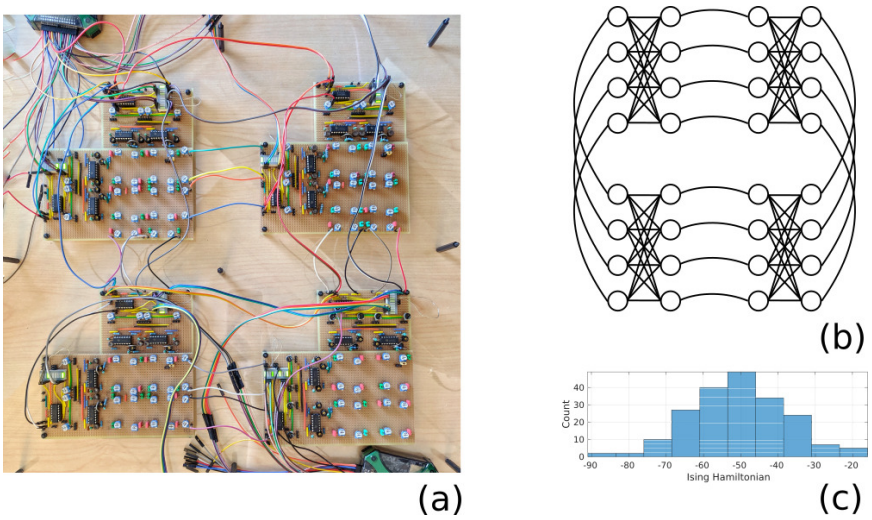



**Fig. 4.9:** A size-32 oscillator-based Ising machine: (a) photo of the implementation on perfboards; (b) illustration of the connectivity ; (c) a typical histogram of the energy values achieved in 200 runs on a random size-32 Ising problem; the lowest energy level is -88 and is achieved once in this case.

an oscillator's response and a reference signal converts the oscillating waveform into a high or low voltage level, indicating if the oscillator's phase is aligned with or opposite to the reference phase. The voltage level can then be picked up by a multichannel logic analyzer. The entire setup is powered by two Digilent Analog Discovery 2 devices, which are portable USB devices that integrate power supplies, logic analyzers and function generators. We have tried random Ising problems by programming each connection with a random polarity using the jumper caps. A typical histogram of the Ising Hamiltonians achieved is shown in Fig. 4.9 (c). Note that because $J_{ij}$s have random polarities, a random solution would have an average energy level of zero. In comparison, the results measured from the hardware are always below 0, and sometimes achieve the global minimum. While such a hand-soldered system is nontrivial to assemble and operate, and its size of 32 cannot be characterized as large scale, it is a useful proof of concept for implementing oscillator-based Ising machines using standard CMOS technologies, and serves as a very solid basis for our future plans to scale the implementations with custom PCBs and custom ICs.

### 4.3.2 MAX-CUT Benchmark Problems

In this section, we demonstrate the efficacy of oscillator-based Ising machines for solving larger-scale MAX-CUT problems. Specifically, we have run simulations on all the problems in a widely used set of MAX-CUT benchmarks known as the G-set [2].[8] Problem sizes range from 800 to 3000.[9] In the experiments, we operated the Ising machine for all

[8]The G-set problems are available for download as set1 at http://www.optsicom.es/maxcut.

[9]G1∼21 are of size 800; G22∼42 are of size 2000; G43∼47, G51∼54 are of size 1000; G48∼50 are of size 3000.

the problems with a single annealing schedule, *i.e.*, we did not tune our Ising machine parameters individually for different problems. Each problem was simulated with 200 random instances. In Tab. 4.1, we list the results and runtime alongside those from several heuristic algorithms developed for MAX-CUT — Scatter Search (SS) [140], CirCut [141], and Variable Neighbourhood Search with Path Relinking (VNSPR) [139].[10] We also list the performances of simulated annealing from a recent study [138], the only one we were able to find that contains results for all the G-set problems.

From Tab. 4.1, we observe that our oscillator-based Ising machine is indeed effective — it finds best-known cut values for 38 out of the 54 problems, 17 of which are even better than those reported in the above literature. Moreover, in the 200 random instances, the best cut is often reached more than once — in average we achieve the maximum 20 times out of 200. The results can in fact be improved further if we tailor the annealing schedule for each problem. But to show the effectiveness and generality of our scheme, we have chosen to use the same annealing schedule for all the problems.

In the annealing schedule we used, the coupling strength $K$ increases linearly, the noise level $K_n$ steps up from 0 to 1, while SYNC's amplitude $K_s$ ramps up and down multiple times. Such a schedule was chosen empirically and appears to work well for most G-set problems. Fig. 4.10 shows the behavior of oscillator phases and the instantaneous cut values under this schedule for solving benchmark problem G1 to its best-known cut size.

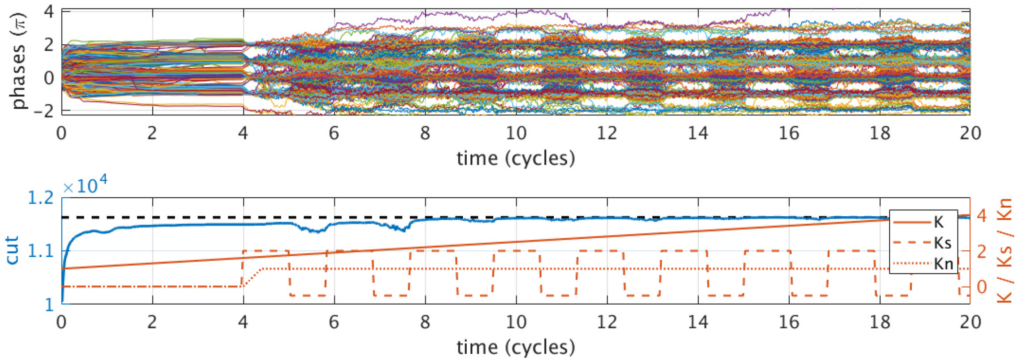



**Fig. 4.10:** Coupled oscillators solving MAX-CUT benchmark problem G1 [2] to its best-known cut size 11624.

The fact that we were using a fixed schedule also indicates that the actual hardware time for the Ising machine to solve all these benchmarks is the same, regardless of problem size and connectivity. Note that in Fig. 4.10, the end time 20 means 20 oscillation cycles, but this end time is predicated on a coupling strength of $K \sim 1$. The actual value of $K$ for each oscillator depends on the PPV function as well as the amplitude of perturbation from other oscillators. As an example, for the LC oscillators we use in Sec. 4.3.1 with

[10] Their results and runtime are available for download at http://www.optsicom.es/maxcut in the "Computational Experiences" section.

| Benchmark | SS | Time | CirCut | Time | VNSPR | Time | SA | Time | OIM | Time |
|---|---|---|---|---|---|---|---|---|---|---|
| G1 | **11624** | 139 | **11624** | 352 | 11621 | 22732 | 11621 | 295 | **11624** | 52.6 |
| G2 | **11620** | 167 | 11617 | 283 | 11615 | 22719 | 11612 | 327 | **11620** | 52.7 |
| G3 | **11622** | 180 | **11622** | 330 | **11622** | 23890 | 11618 | 295 | **11622** | 52.4 |
| G4 | **11646** | 194 | 11641 | 524 | 11600 | 24050 | 11644 | 294 | **11646** | 52.7 |
| G5 | **11631** | 205 | 11627 | 1128 | 11598 | 23134 | 11628 | 300 | **11631** | 52.6 |
| G6 | 2165 | 176 | **2178** | 947 | 2102 | 18215 | **2178** | 247 | **2178** | 52.8 |
| G7 | 1982 | 176 | 2003 | 867 | 1906 | 17716 | **2006** | 205 | 2000 | 52.9 |
| G8 | 1986 | 195 | 2003 | 931 | 1908 | 19334 | **2005** | 206 | 2004 | 52.8 |
| G9 | 2040 | 158 | 2048 | 943 | 1998 | 15225 | **2054** | 206 | **2054** | 52.6 |
| G10 | 1993 | 210 | 1994 | 881 | 1910 | 16269 | 1999 | 205 | **2000** | 52.9 |
| G11 | 562 | 172 | 560 | 74 | **564** | 10084 | **564** | 189 | **564** | 6.7 |
| G12 | 552 | 242 | 552 | 58 | **556** | 10852 | 554 | 189 | **556** | 6.3 |
| G13 | 578 | 228 | 574 | 62 | 580 | 10749 | 580 | 195 | **582** | 6.4 |
| G14 | 3060 | 187 | 3058 | 128 | 3055 | 16734 | **3063** | 252 | 3061 | 14.6 |
| G15 | **3049** | 143 | **3049** | 155 | 3043 | 17184 | **3049** | 220 | **3049** | 16.1 |
| G16 | 3045 | 162 | 3045 | 142 | 3043 | 16562 | 3050 | 219 | **3052** | 14.5 |
| G17 | 3043 | 313 | 3037 | 366 | 3030 | 18555 | 3045 | 219 | **3046** | 14.6 |
| G18 | 988 | 174 | 978 | 497 | 916 | 12578 | **990** | 235 | **990** | 14.7 |
| G19 | 903 | 128 | 888 | 507 | 836 | 14546 | 904 | 196 | **906** | 14.5 |
| G20 | **941** | 191 | **941** | 503 | 900 | 13326 | **941** | 195 | **941** | 14.7 |
| G21 | 930 | 233 | **931** | 524 | 902 | 12885 | 927 | 195 | **931** | 14.6 |
| G22 | 13346 | 1336 | 13346 | 493 | 13295 | 197654 | 13158 | 295 | **13356** | 58.7 |
| G23 | 13317 | 1022 | 13317 | 457 | 13290 | 193707 | 13116 | 288 | **13333** | 58.6 |
| G24 | 13303 | 1191 | 13314 | 521 | 13276 | 195749 | 13125 | 289 | **13329** | 59.0 |
| G25 | 13320 | 1299 | **13326** | 1600 | 12298 | 212563 | 13119 | 316 | **13326** | 58.7 |
| G26 | 13294 | 1415 | **13314** | 1569 | 12290 | 228969 | 13098 | 289 | 13313 | 58.9 |
| G27 | 3318 | 1438 | 3306 | 1456 | 3296 | 35652 | **3341** | 214 | 3323 | 59.0 |
| G28 | 3285 | 1314 | 3260 | 1543 | 3220 | 38655 | **3298** | 252 | 3285 | 61.2 |
| G29 | 3389 | 1266 | 3376 | 1512 | 3303 | 33695 | 3394 | 214 | **3396** | 58.9 |
| G30 | 3403 | 1196 | 3385 | 1463 | 3320 | 34458 | **3412** | 215 | 3402 | 59.0 |
| G31 | 3288 | 1336 | 3285 | 1448 | 3202 | 36658 | **3309** | 214 | 3296 | 59.1 |
| G32 | 1398 | 901 | 1390 | 221 | 1396 | 82345 | **1410** | 194 | 1402 | 17.5 |
| G33 | 1362 | 926 | 1360 | 198 | **1376** | 76282 | **1376** | 194 | 1374 | 15.9 |
| G34 | 1364 | 950 | 1368 | 237 | 1372 | 79406 | **1382** | 194 | 1374 | 15.9 |
| G35 | 7668 | 1258 | 7670 | 440 | 7635 | 167221 | 7485 | 263 | **7675** | 37.1 |
| G36 | 7660 | 1392 | 7660 | 400 | 7632 | 167203 | 7473 | 265 | **7663** | 37.6 |
| G37 | 7664 | 1387 | 7666 | 382 | 7643 | 170786 | 7484 | 288 | **7679** | 37.8 |
| G38 | **7681** | 1012 | 7646 | 1189 | 7602 | 178570 | 7479 | 264 | 7679 | 37.7 |
| G39 | 2393 | 1311 | 2395 | 852 | 2303 | 42584 | **2405** | 209 | 2404 | 37.2 |
| G40 | 2374 | 1166 | 2387 | 901 | 2302 | 39549 | 2378 | 208 | **2389** | 38.1 |
| G41 | 2386 | 1017 | 2398 | 942 | 2298 | 40025 | **2405** | 208 | 2401 | 37.8 |
| G42 | 2457 | 1458 | **2469** | 875 | 2390 | 41255 | 2465 | 210 | **2469** | 37.3 |
| G43 | 6656 | 406 | 6656 | 213 | 6659 | 35324 | 6658 | 245 | **6660** | 29.1 |
| G44 | **6648** | 356 | 6643 | 192 | 6642 | 34519 | 6646 | 241 | **6648** | 29.2 |
| G45 | 6642 | 354 | 6652 | 210 | 6646 | 34179 | 6652 | 241 | **6653** | 29.1 |
| G46 | 6634 | 498 | 6645 | 639 | 6630 | 38854 | 6647 | 245 | **6649** | 29.1 |
| G47 | 6649 | 359 | **6656** | 633 | 6640 | 36587 | 6652 | 242 | **6656** | 29.1 |
| G48 | **6000** | 20 | **6000** | 119 | **6000** | 64713 | **6000** | 210 | **6000** | 23.2 |
| G49 | **6000** | 35 | **6000** | 134 | **6000** | 64749 | **6000** | 210 | **6000** | 23.2 |
| G50 | **5880** | 27 | **5880** | 231 | **5880** | 147132 | 5858 | 211 | 5874 | 25.6 |
| G51 | **3846** | 513 | 3837 | 497 | 3808 | 89966 | 3841 | 234 | **3846** | 18.4 |
| G52 | **3849** | 551 | 3833 | 507 | 3816 | 95985 | 3845 | 228 | 3848 | 18.4 |
| G53 | **3846** | 424 | 3842 | 503 | 3802 | 92459 | 3845 | 230 | **3846** | 18.4 |
| G54 | 3846 | 429 | 3842 | 524 | 3820 | 98458 | 3845 | 228 | **3850** | 18.5 |

**Table 4.1:** Results of oscillator-based Ising machines run on MAX-CUT benchmarks in the G-set, compared with several heuristic algorithms. Time reported in this table is for a single run.

100k resistive coupling, $K \approx 0.02$. This indicates that it takes less than 100 cycles for the oscillators to synchronize in phase, which is consistent with measurements. For such a coupled LC oscillator network, a hardware time of 20 in Fig. 4.10 represents approximately 2000 cycles of oscillation; for 5MHz oscillators that takes 0.4ms. If we use GHz nano-oscillators, the computation time can be well within a microsecond. In comparison, the runtime of the several heuristic algorithms listed in Tab. 4.1, even with faster CPUs and parallel implementations in the future, is unlikely to ever drop to this range.

As the hardware time is fixed, the runtime we report in Tab. 4.1 for our Ising machines is the time for simulating the SDEs of coupled oscillators on CPUs. While we list runtime results for each algorithm in Tab. 4.1, note that they come from different sources and are measured on different platforms. Results for SS, CirCut and VNSPR were obtained from Dual Intel Xeon at 3.06GHz with 3.2GB of RAM; SA was run on Intel Xeon E3-1245v2 at 3.4GHz with 32GB of RAM [138]. To make the results generally comparable, we ran our simulations on a modest personal desktop with Intel Xeon E5-1603v3 at 2.8GHz with 16GB of RAM. Even so, it came as a nice surprise to us that even by simulating SDEs we were able to solve the benchmarks efficiently. Another notable feature of our method is that unlike other algorithms, SDE simulation does not know about the Ising Hamiltonian or cut value — it never needs to evaluate the energy function or relative energy changes, which are implicit in the dynamics of differential equations, yet it proves effective and fast.

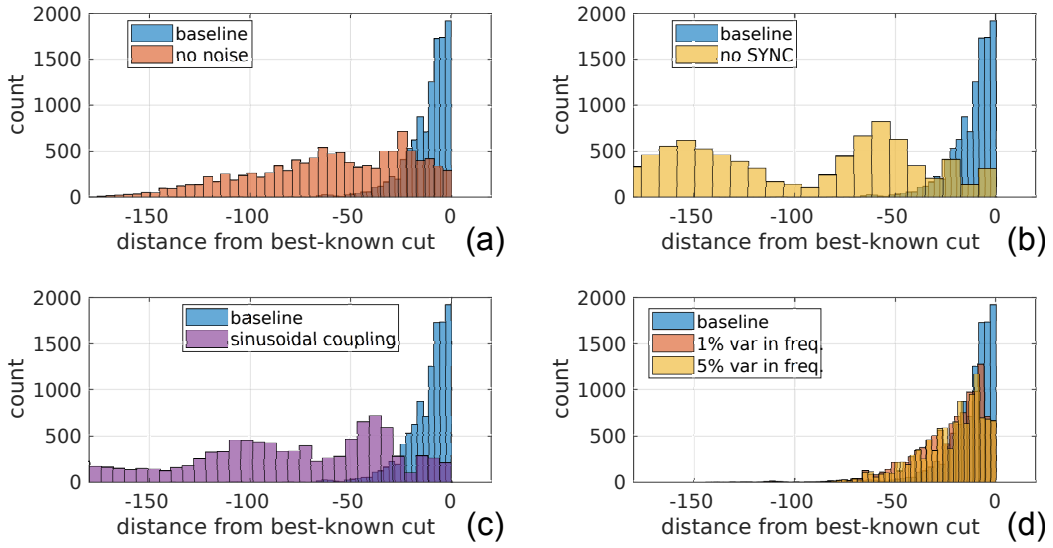



**Fig. 4.11:** Histograms of the cut values achieved by several variants of the Ising machine, compared with the baseline results used in Tab. 4.1.

We also ran more computational experiments on the G-set benchmarks in order to study the mechanism of oscillator-based Ising machines. We created several variants of the Ising machine used above by removing different components in its operation. For each variant, we re-ran 200 random instances for each of the 54 benchmarks, generating 10800 cut values.

In Fig. 4.11, we compare the quality of these cut values with results from the unaltered Ising machine by plotting histograms of the distances of the cut values to their respective maxima. In the first variant, we removed noise from the model by setting $K_n \equiv 0$. The solutions become considerably worse, confirming that noise helps the coupled oscillator system settle to lower energy states.

In the next variant, we removed SYNC by setting $K_s \equiv 0$. Without SYNC, the system becomes a simple coupled oscillator system with phases that take a continuum of values, as discussed in Sec. 4.2.1. The settled analog values of the phases that were then thresholded to 0 or $\pi$ to correspond to Ising spins. As shown in Fig. 4.11, the results become significantly worse; indeed, none of the best-known results were reached. This indicates that the SYNC signal and the mechanism of SHIL we introduce to the coupled oscillator networks are indeed essential for them to operate as Ising machines.

Our baseline Ising machine actually uses a smoothed square function $\tanh(\sin(.))$ for the coupling, as opposed to the $\sin(.)$ used in the original Kuramoto model. This changes the $\cos(.)$ term in the energy function (4.7) to a triangle function. Such a change appears to give better results than the original, as shown in Fig. 4.11 (c). The change requires designing oscillators with special PPV shapes and waveforms such that their cross-correlation is a square wave, which is not difficult in practice based on our derivation in Sec. 2.2. As an example, rotary traveling wave oscillators naturally have square PPVs. Ring oscillators can also be designed with various PPVs and waveforms by sizing each stage individually. We cannot say definitively that the square function we have used is optimal for Ising solution performance, but the significant improvement over sinusoidal coupling functions indicates that a fruitful direction for further exploration may be to look beyond the original Kuramoto model for oscillator-based computing.

The last variant we report here added variability to the natural frequencies of the oscillators, as in (4.20). We assigned Gaussian random variables to $\omega_i$s with $\omega^*$ as the mean, and 0.01 (1%) and 0.05 (5%), respectively, as the standard deviations for two separate runs. From Fig. 4.11 (d), we observe that even with such non-trivial spread in the natural frequencies of oscillators, the performance is affected very little.

Finally, we conducted a preliminary study of the scaling of the time taken by the Ising machine to reach good solutions as problem sizes increase. As the G-set benchmarks have only a few sizes (800, 1000, 2000 and 3000), we used the program (named `rudy`) that generated them to create more problems of various sizes. All generated problems used random graphs with 10% connectivity and $\pm 1$ coupling coefficients. We simulated all of them, each for 200 instances, with fixed parameters $K = 1$, $K_s = 0.1$, $K_n = 0.01$, and show all their Ising Hamiltonians over time in Fig. 4.12. Much to our surprise, the speed in which the values settle appears almost constant, regardless of the problem size. While this does not necessarily mean they all converge to the global optima within the same time, this preliminary study is encouraging as it confirms the massively parallel nature of the system. For larger Ising problems, our Ising machine only needs to scale linearly in hardware size

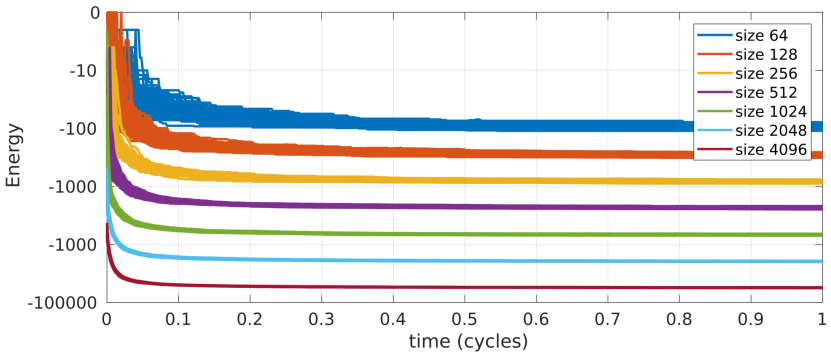



**Fig. 4.12:** Speed of energy minimization for problems of different sizes.

with the number of spins, but does not necessarily require much more time to reach a solution.

### 4.3.3 A Graph Coloring Example

As mentioned in Sec. 4.1, many problems other than MAX-CUT can be mapped to the Ising model [122] and solved by an oscillator-based Ising machine. Here we show an example of a graph coloring problem — assigning four colors to the 51 states (including a federal district) of America such that no two adjacent states have the same color.

Each state is represented as a vertex in the graph. When two states are adjacent, there is an edge in the graph that connects the corresponding vertices. For every vertex $i$, we assign four spins — $s_{iR}$, $s_{iG}$, $s_{iB}$ and $s_{iY}$ to represent its coloring scheme; when only one of them is $+1$, the vertex is successfully colored as either red, green, blue or yellow. Then we write an energy function $H$ associated with these $4\times 51 = 204$ spins as follows:

$$H = \sum_{i}^{n}(2+s_{iR}+s_{iG}+s_{iB}+s_{iY})^2 + \sum_{(i,j)\in\mathbb{E}}^{n_{\mathbb{E}}}\left[(1+s_{iR})(1+s_{jR})+(1+s_{iG})(1+s_{jG})+(1+s_{iB})(1+s_{jB})+(1+s_{iY})(1+s_{jY})\right], \tag{4.27}$$

where $n = 51$ is the number of vertices, $\mathbb{E}$ represents the edge set, $n_{\mathbb{E}}$ is the number of edges and in this case equal to 220.[11]

The first term of $H$ is a sum of squares never less than zero; it reaches zero only when $\{s_{iR},\ s_{iG},\ s_{iB},\ s_{iY}\}$ contains three $-1$s and one $+1$ for every $i$, *i.e.*, each state has a unique color. The latter term is also a sum that is always greater than or equal to zero, as each spin can only take a value in $\{-1,\ +1\}$; it is zero when $s_{iX} = s_{jX} = +1$ never occurs

[11] Hawaii and Alaska are considered adjacent such that their colors will be different in the map.

for any edge connecting $i$ and $j$, and for any color $X \in \{R,\ G,\ B,\ Y\}$, *i.e.*, adjacent states do not share the same color. Therefore, when $H$ reaches its minimum value 0, the spin configuration represents a valid coloring scheme — following the indices of the $+1$ spins $\{i,\ X \mid s_{iX} = +1\}$, we can then assign color $X$ to state $i$.

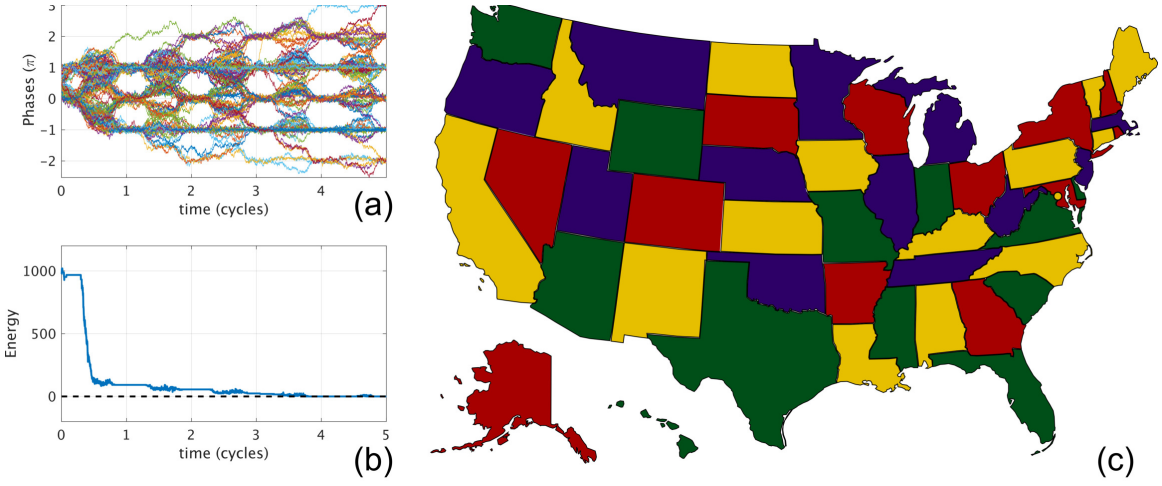



**Fig. 4.13:** Coupled oscillators coloring the states in the US map: (a) phases of oscillators evolve over time; (b) energy function (4.27) decreases during the process; (c) the resulting US map coloring scheme.

Note that when expanding the sum of squares in (4.27), we can use the fact $s_{iX}^2 \equiv 1$ to eliminate the square terms. This means $H$ contains only products of two spins — modelled by $J_{ij}$s, and self terms — modelled by $h_i$. These Ising coefficients can then be used to determine the couplings in an oscillator-based Ising machine.

We simulated these 204 coupled oscillators and show the results in Fig. 4.13. In the simulation, we kept $K$ and $K_n$ constant while ramping $K_s$ up and down 5 times. We found the Ising machine to be effective with this schedule as it could color the map successfully in more than 50% of the random trials and return many different valid coloring schemes.

### 4.3.4 Hardware Prototypes

In Sec. 4.3.1 we have presented two hardware prototypes: a size-8 oscillator-based Ising machine on a breadboard (OIM8) and a size-32 one on interconnected perfboards (OIM32) in the context of solving small-sized MAX-CUT problems Here in this section we present two more: OIM64 (Fig. 4.14) and OIM240 (Fig. 4.14).

Other than the differences in system size (number of oscillator spins), they have a few key differences from OIM8 and OIM32.

- OIM64 and OIM240 are built on printed circuit broads (PCBs) as opposed to breadboards or perfboards.

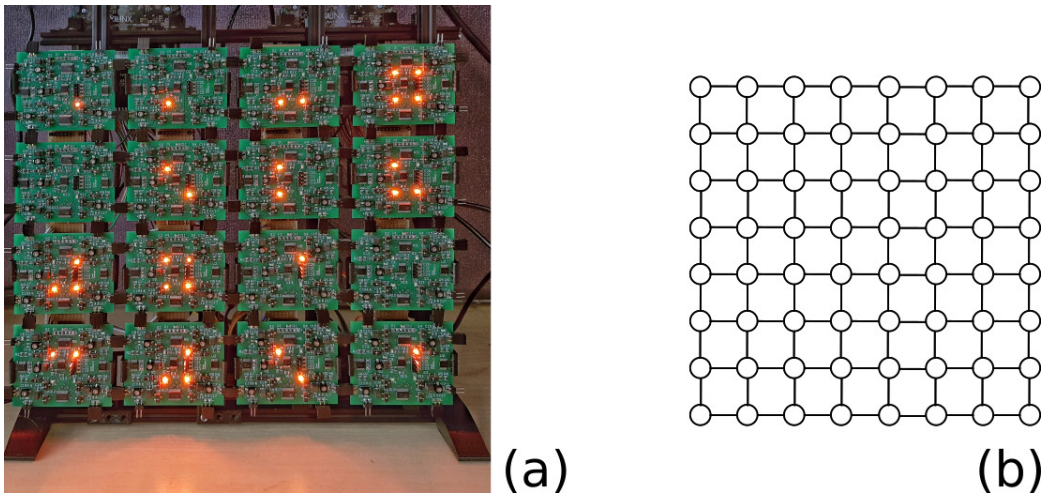



**Fig. 4.14:** OIM64 Prototype. (a) photo; (b) diagram of connectivity.

- Instead of using fixed resistors or rotary potentiometers that require a lot of manual operation, Both OIM64 and OIM240 use digital potentiometers implemented using AD5206 ICs. Each IC supplies 6 channels of potentiometers, each with a 8-bit accuracy and can be programmed using serial peripheral interface (SPI) communication to the IC.
- Because AD5206 ICs (and almost all other digital potentiometers) are designed primarily for audio processing and do not have multi-MHz bandwidth, we reduced oscillator frequency from 5MHz to 1MHz.

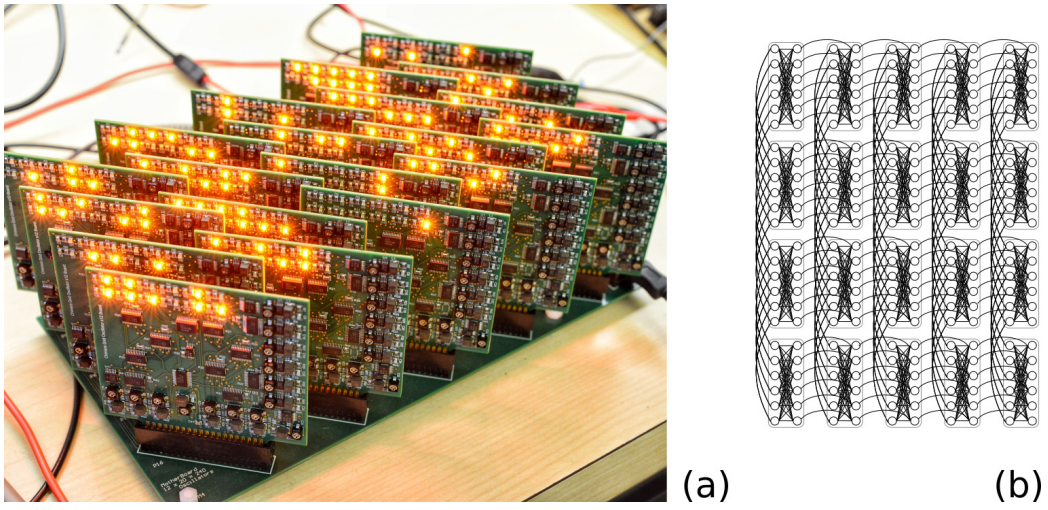



**Fig. 4.15:** OIM240 Prototype. (a) photo; (b) diagram of connectivity.

OIM64 connects 64 oscillators in a 8-by-8 2-D toroidal grid, and has 192 potentiometers as couplings. Each coupling consists of one channel of AD5206 for setting its resistance,

| benchmark | seed | energy | cut |
|---|---|---|---|
| J64_01.rud | 8001 | -88 | 40 |
| J64_02.rud | 8002 | -82 | 42 |
| J64_03.rud | 8003 | -92 | 46 |
| J64_04.rud | 8004 | -98 | 40 |
| J64_05.rud | 8005 | -82 | 44 |
| J64_06.rud | 8006 | -90 | 52 |
| J64_07.rud | 8007 | -84 | 48 |
| J64_08.rud | 8008 | -94 | 56 |
| J64_09.rud | 8009 | -88 | 48 |
| J64_10.rud | 8010 | -86 | 44 |

**Table 4.2:** Best results measured from OIM64 on size-64 rudy-generated benchmarks. Example rudy command: `rudy -toroidal_grid_2D 8 8 -random 0 1 8001 -times 2 -plus -1 > J64_01.rud`.

and a single pole double throw (SPDT) on-off-on switch for setting its polarity.[12] Even though it turned out rather inconvenient to "program" OIM64 to encode problems due to the use of physical switches, we tried it on 10 randomly generated toroidal Ising grid instances, achieving the global optimum for each one within the first 100 samples of solutions. As the benchmarks are generated by `rudy`, they are readily reproducible. Tab. 4.2 lists the results measured from OIM64, and the command that generates the benchmarks.

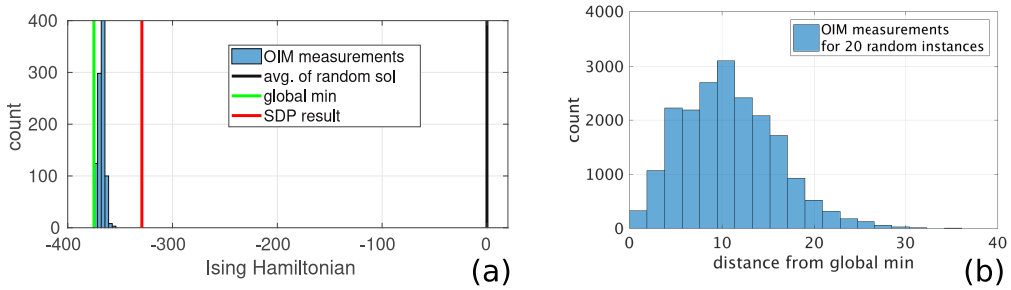



**Fig. 4.16:** Results from OIM240: (a) energy levels of 1000 measured solutions from OIM240, on a random instance of size-240 Ising problem; (b) energy levels for 20 such random instances (showing the distances to their respective global minimum).

In OIM240, we have improved the design to use the position of the potentiometer wiper to switch polarity, thus eliminating the use of switches and making the coupling truly programmable. On each PCB, we implemented 12 oscillators with a denser connectivity; 20 such PCBs were plugged into a motherboard through edge connectors, and interconnected in a 4-by-5 toroidal grid, implementing a total of 240 oscillators with 1200 couplings.

[12] When the switch is in the middle "off" state, the resistor is disconnected and the coupling coefficient becomes zero.

The motherboard also distributes CLK, data lines, address lines for programming the 200 AD5206 ICs and for reading oscillator states, all controlled by an Arduino module on the motherboard that communicates with a PC through USB. When operating OIM240, we flip on the supply digitally, wait 1ms for oscillators to synchronize, then read back the solution. Even with all the overhead from serial reading, solutions can be read back every 5ms. OIM240's operation consumes $\sim$5W of power for all the oscillators and peripheral circuitry, excluding only the LEDs.

We have tested OIM240 with many randomly generated Ising problems (with each of the 1200 couplings randomly chosen from 0, -1, +1). As `rudy` does not support OIM240's chimera connectivity, we wrote our own random graph generator for the test cases. A typical histogram for the energy levels of the measured solutions is shown in Fig. 4.16 (a). Note that a random (trivial) solution has an energy around 0, whereas the best polynomial-time algorithm (based on semidefinite programming SDP) guarantees to achieve 87.8% of the global optimum. In comparison, results from OIM240 center around a very low energy, and achieve the global optimum multiple times. We performed the same measurements for 20 different random Ising problems, with the distances of solutions from their respective global optima[13] shown in Fig. 4.16 (b). The fact that OIM240 is finding highly non-trivial solutions indicates that it indeed physically implements a working Ising machine.

[13]We ran simulated annealing for a long time (1min) and for multiple times, then treated the best results as global optima.

# Chapter 5

# Design Tools: Berkeley Model and Algorithm Prototyping Platform

Novel computing paradigms call for modern design tools. To use any new materials/devices or circuits/systems for computing (such as those listed in Sec. 1.1), we need accurate device models and advanced simulation algorithms. Good design tools should allow us to prototype both of them easily and quickly.

But as we show in Sec. 5.1, existing simulation platforms are generally not suited for this purpose. This situation has motivated us to develop our own design tools — MAPP: the Berkeley Model and Algorithm Prototyping Platform [142, 143]. Sec. 5.2 introduces the design and features of MAPP. Then in Sec. 5.3 and Sec. 5.4, we use examples to illustrate MAPP's capability for prototyping device models and simulation algorithms respectively.

In particular, MAPP has greatly facilitated our work in device modelling. With MAPP's flexible device development flow and its powerful circuit/system analyses, we are able to develop robust models and valuable modelling methodologies. Prominent examples include the models for devices with hysteresis (memristors, ESD protection devices, *etc.*) [95, 96, 144] and devices from multiple physical domains (optical, spintronic, mechanical, *etc.*) [145, 146]. These examples are illustrated in Sec. 5.3.

MAPP has been an enabling tool for our work on oscillator-based computing systems [147]. The study and design of such systems (Chapter 3 and Chapter 4) rely on accurate models for multi-domain oscillators (*e.g.*, the electrical and mechanical ones used in Sec. 3.3) and advanced algorithms for circuit/oscillator analysis (*e.g.*, PPV extraction, Adlerization, coupled oscillator simulation, *etc.* have been used throughout this dissertation). In Sec. 5.5, we discuss in more detail how we have used MAPP in our work on oscillators, and how it can continue facilitating the research and deployment of oscillator-based computing paradigms in the future.

## 5.1 Existing Open-source Simulators and Motivation for MAPP

Perhaps the most famous open-source simulator is the Berkeley SPICE [148, 149]. SPICE and its derivatives, *e.g.*, ngspice [150], are widely used and have long been the standard platforms for modelling and simulation. But they are not well suited for quick and convenient prototyping, primarily because of the outdated code structuring. In particular, simulation algorithms in Berkeley SPICE are implemented in, and specialized for, every device model. Such structuring makes it difficult to insert new devices or algorithms.

As an example, an excerpt from SPICE3's implementation of a diode model is shown in Fig. 5.1. The code shown has almost no direct relation to the diode's equations; it relates to the implementations of sensitivity AC, transient, *etc.*, analyses written inside the device model. In SPICE3, the complete diode implementation involves 27 different files, with 2704 lines of code in all. To add a new device model into SPICE and its derivatives, the model developer needs to implement many analyses (such as DC operating point, DC sweep, AC, transient (including time step control and convergence aids)), and tailor them based on the device equations. This requires expert-level knowledge not only on the various analyses, but also on the internal code structuring of SPICE. On the other hand, writing a new simulation algorithm requires deep knowledge on the device equations too; a researcher prototyping a new analysis will need to implement it in and tailor it for all the devices.

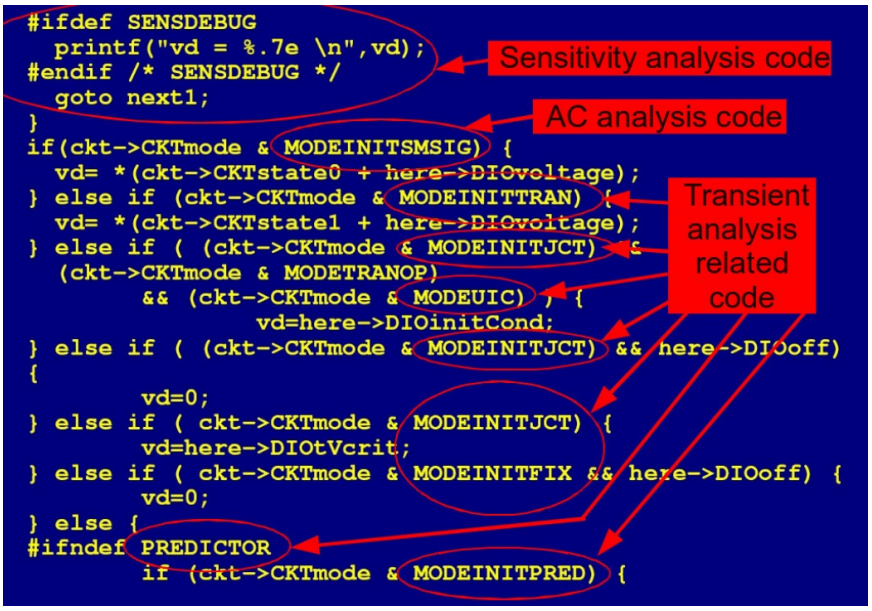



**Fig. 5.1:** Excerpt from SPICE3's dioload.c, illustrating how every device contains code for every analysis.

While modern open-source post-SPICE circuit simulators, *e.g.*, Gnucap [151] and Qucs [152], have many advantages over SPICE and its derivatives, especially in code readability and documentation, they continue to implement algorithms in devices, hence adding either remains challenging. Xyce [153], a modern open source simulator, allows device models

to be nearly independent of analysis types, alleviating the difficulty of developing models and algorithms to a great extent. However, code development in Xyce requires considerable facility with C++ programming.

This situation has long hindered research in modelling and simulation — the barrier to entry for incorporating new devices or analyses is so high that few researchers are capable of performing these tasks effectively. New ideas are often dropped simply because they cannot be prototyped in a reasonable time using available open source simulators. This long-standing barrier has been the main motivation behind our work on MAPP.

Another motivation for MAPP is to model and simulate multiphysics devices and systems, for which standard electronic simulators available today are not suitable. SPICE and its derivatives were designed primitively for electronics. Specifically, the circuit equations are constructed in each device through "stamping", based on hard-coded Modified Nodal Analysis (MNA) [55] and current right-hand-side (RHS) formulation [154]. Post-SPICE circuit simulators, such as Gnucap, Qucs, and Xyce, still carry with them many of the same circuit modelling conventions, such as MNA, making them ill-equipped for multiphysics.

As a result, much research effort has been devoted to adapting open-source SPICE-like simulators to support non-electronic systems, for MEMS [155], chemical reactions [156], neural systems [157], river networks [158], *etc.*. These techniques essentially implement "wrappers" around electronic simulators for other physical domains, and only support the domain for which they are designed. Similarly, for commercial simulators, Verilog-A provides an industry standard "wrapper" for modelling non-electronic systems. However, although Verilog-A models can be written with variables from different "natures" and disciplines [159], these variables have to be either potentials or flows, corresponding to voltages and currents in the underlying circuit formulation in simulators, which still internally use hard-coded MNA and circuit laws (KCLs/KVLs). For systems that don't obey circuit laws, *e.g.*, optoelectronic [160] and spintronic [3] systems, models are often not intuitive to write and require cumbersome connections in netlists. This difficulty is another motivation behind MAPP.

## 5.2 Features of MAPP

From above, the primary goal of MAPP is to ease the process of developing new device models and simulation algorithms, especially for those who do not have an extensive background in compact modelling or experience coding algorithms in simulators. Towards this end, we have chosen to implement MAPP entirely in MATLAB®.[1] MATLAB® is widely used today in scientific and engineering communities; its simple, mathematics-based syntax makes programming accessible to a broad range of users. In addition, MATLAB® is interactive, as well as interpreted (*i.e.*, there is no need for compilation), which makes it

[1] Later developments on MAPP have led to its reimplementation in python and C++. We intend to release their implementations to be public too. In this section, we focus on our MATLAB®-based implementation of MAPP.

well suited for quick prototyping and debugging. It has built-in support for vectors/matrices (including sparse matrices) and comes with an exceptionally rich set of mathematical objects and functions in linear algebra, statistics, Fourier analysis, optimization, bioinformatics, *etc.*, all useful for compact modelling and simulation. MATLAB® also offers flexible, easy-to-use graphics/visualization facilities which are valuable when exploring new devices and analyses.

Another feature of MAPP is its code structuring [142, 161], which differs markedly from that of SPICE. The modular code structure is what makes prototyping models and algorithms fast and easy. As shown in Fig. 5.2, the structure of MAPP centers around a mathematical abstraction — the Differential Algebraic Equation (DAE) [55]. DAE is well suited for describing virtually any continuous-time dynamical systems; we have been using it throughout this dissertation in the modelling of oscillators. The use of DAEs also enables MAPP to model and simulate devices and systems from domains beyond just the electrical in a natural way. Device equations are specified in a MATLAB®-based format (ModSpec [162]). With the DAE concept separating device equations and analysis algorithms, MAPP's ModSpec device models are unaware of what simulation algorithms there are that may use them. This simplifies and speeds up the task of device model prototyping in MAPP.

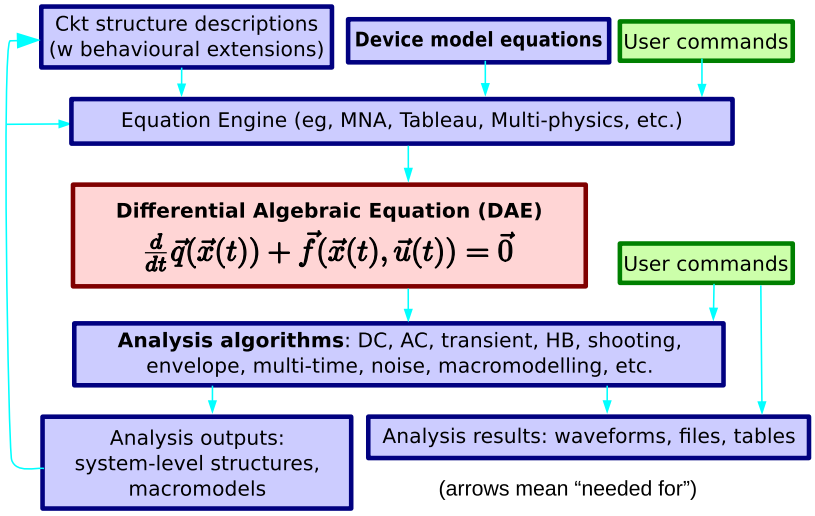



**Fig. 5.2:** Components of MAPP.

DAEs are set up by equation engines, a concept unique to MAPP. An equation engine combines network connectivity information (*e.g.*, from a circuit netlist) with device model equations in ModSpec to produce system-level DAEs. MAPP's simulation algorithms are aware only of DAE objects; they work by calling DAE accessor functions (collectively called DAEAPI). This structuring enables developers to add new algorithms knowing only the generic format of DAEs, without having to look into the details of device implementation or equation formulation.

For several years before its public release, MAPP has been used internally in our group

and has greatly facilitated our own research. For instance, we developed, implemented and validated a new algorithm for distortion computation [163] using MAPP; this was done by a fresh graduate student, new to the field, in about three weeks. By way of comparison, the much more limited distortion capability in Berkeley SPICE3 [149] had taken over a year to implement — by a graduate student already familiar with coding within SPICE3. Implementing our new algorithm would have involved making so many changes to SPICE3, and taken so much time and effort, that we would not even have tried; the idea would have been lost. Not only has MAPP been useful for trying new ideas, it has also served as a key vehicle for meaningful collaboration with other research groups. Furthermore, it has been helpful for teaching simulation and modelling concepts, with preliminary versions used in classes over several years.

MAPP has been released publicly in open source form [164], primarily under the GNU Public License, with alternative licensing models also supported. The current release of MAPP contains common electrical devices and standard simulation algorithms, including DC, AC and transient analyses. Many more capabilities, including multi-physics device and system modelling features, and additional analyses such as shooting, harmonic balance, homotopy, stationary noise analysis, parameter sensitivity analysis, per-element distortion analysis, model order reduction based on moment matching and Krylov subspace methods, *etc.* [55], have already been prototyped in MAPP and will also be made available under MAPP's open source license. MAPP comes with a suite of examples at the device, system and analysis levels. It leverages MATLAB®'s help system to help new users get started, and to provide more advanced users information about internal structuring and available functions. It also includes an automatic testing system, designed to facilitate development by quickly detecting problems as code is written or changed. All these features help make MAPP ideal for rapid prototyping of device models and simulation algorithms.

## 5.3 Compact Modelling in MAPP

Fig. 5.3 depicts the device model prototyping flow in MAPP. We use a tunnel diode example below to explain the steps that constitute this flow, illustrating its advantages and novelties along the way.

Step 1.1: *Writing the model in ModSpec*:

Writing a model will normally start by specifying model equations in the ModSpec format [162], which we illustrate using a tunnel diode model. Tunnel diodes [165] are a type of two-terminal semiconductor device with I/V characteristics similar to the blue curve in Fig. 5.5. We name the model's two terminals $p$ and $n$, as shown in Fig. 5.4. Associated with these terminals are two electrical I/O properties: a branch voltage $vpn$ and a branch current $ipn$. In their simplest form, their relationship can be written as [165]

$$i_{pn} = \frac{d}{dt}\left(C \cdot v_{pn}\right) + I_{\text{diode}} + I_{\text{tunnel}} + I_{\text{excess}}, \tag{5.1}$$

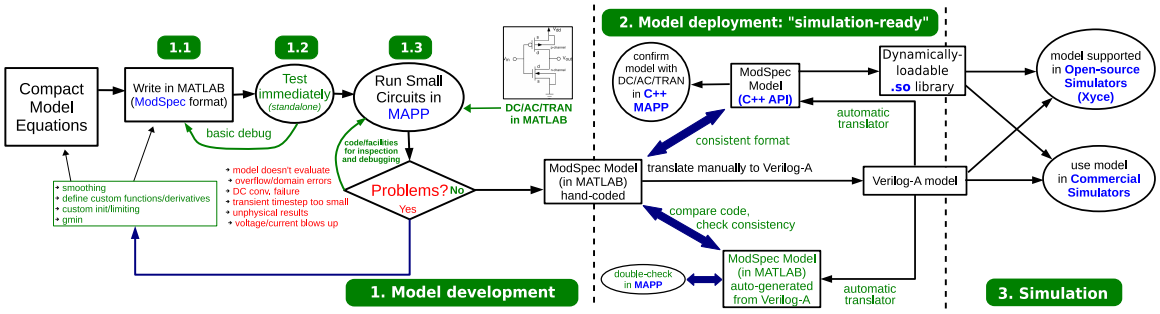



**Fig. 5.3:** Device model prototyping flow in MAPP.

where $C \cdot v_{pn}$ models the charge between the terminals. $I_{\text{diode}}$, $I_{\text{tunnel}}$ and $I_{\text{excess}}$ are the regular diode current, tunnelling current and additional parasitic tunnelling current terms, respectively:

$$I_{\text{diode}} = I_s \cdot \exp(\frac{v_{pn}}{V_t} - 1), \tag{5.2}$$

$$I_{\text{tunnel}} = \frac{I_p}{V_p} \cdot v_{pn} \cdot \exp(-\frac{v_{pn} - V_p}{V_p}), \tag{5.3}$$

$$I_{\text{excess}} = \frac{I_v}{V_v} \cdot v_{pn} \cdot \exp(v_{pn} - V_v). \tag{5.4}$$

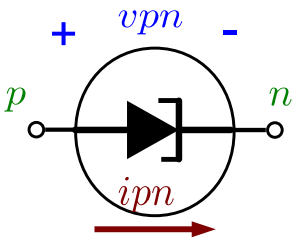



**Fig. 5.4:** Tunnel diode schematic.

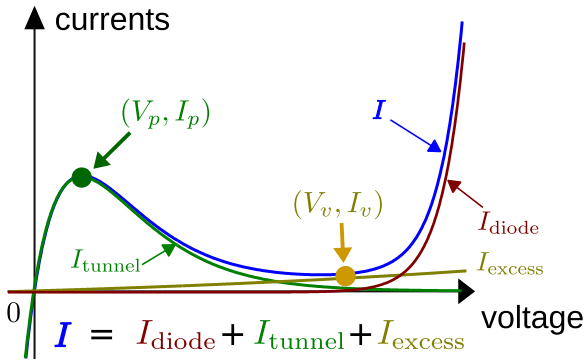



**Fig. 5.5:** Tunnel diode I/V curve.

The above equations involve the model parameters $C$, $I_s$, $V_t$, $V_p$, $I_v$ and $V_v$, which determine the shape of the tunnel diode's characteristic curve and its dynamics. (5.1) is a nonlinear differential equation, with one of its I/Os, namely $ipn$, expressed explicitly – such differential equations are at the core of all device compact models. ModSpec supports the following

general system of equations for devices [162]:

$$\vec{z} = \frac{d}{dt}\vec{q}_e(\vec{x},\vec{y}) + \vec{f}_e(\vec{x},\vec{y},\vec{u}), \tag{5.5}$$

$$0 = \frac{d}{dt}\vec{q}_i(\vec{x},\vec{y}) + \vec{f}_i(\vec{x},\vec{y},\vec{u}). \tag{5.6}$$

The vector quantities $\vec{x}$ and $\vec{z}$ contain the device's terminal I/Os: $\vec{z}$ comprises those I/Os that can be expressed explicitly ($ipn$ for our tunnel diode example), while $\vec{x}$ comprises those that cannot ($vpn$ for the tunnel diode). $\vec{y}$ contains the model's internal unknowns (*e.g.*, internal nodes), while $\vec{u}$ provides a mechanism for specifying time-varying inputs within the device (*e.g.*, as in independent voltage or current sources). (Our tunnel diode example has no entries in $\vec{y}$ and $\vec{u}$.) The functions $\vec{q}_e$, $\vec{f}_e$, $\vec{q}_i$ and $\vec{f}_i$ define the differential and algebraic parts of the model's explicit and implicit equations. The tunnel diode (5.1) can be expressed in ModSpec as

$$\begin{aligned} \vec{f}e(\vec{x},\vec{y},\vec{u}) &= I_{\text{diode}}(\vec{x}) + I_{\text{tunnel}}(\vec{x}) + I_{\text{excess}}(\vec{x}), \\ \vec{q}e(\vec{x},\vec{y}) &= C\cdot\vec{x}, \\ \vec{f}i(\vec{x},\vec{y},\vec{u}) &= [],\ \ \vec{q}i(\vec{x},\vec{y}) = [], \end{aligned} \tag{5.7}$$

with $\vec{x} = [vpn]$, $\vec{y} = []$, $\vec{z} = [ipn]$, $\vec{u} = []$. Issuing the command "`help ModSpec_concepts`" within MAPP provides more detailed explanations of these concepts.

ModSpec objects in MAPP are simply MATLAB® structures that contain a number of datum and function-handle fields. "`help ModSpecAPI`" in MAPP provides detailed documentation of these fields. Writing a ModSpec device model involves providing basic model information (*e.g.*, the number of terminals, internal nodes, which I/Os are explicitly available, parameter names/values, *etc.*) as data fields, and writing the model functions $\vec{q}_e$, $\vec{f}_e$, $\vec{q}_i$, $\vec{f}_i$ using standard MATLAB® syntax. For example, the tunnel diode model above is described with the following ModSpec code:

```
function MOD = Tunnel_Diode_ModSpec()
  MOD = ee_model();
  MOD = add_to_ee_model(MOD,'terminals', {'p', 'n'});
  MOD = add_to_ee_model(MOD,'explicit_outs', {'ipn'});
  MOD = add_to_ee_model(MOD,'parm',{'Is',1e-12,'Vt',0.025});
  MOD = add_to_ee_model(MOD,'parm',{'Ip',3e-5,'Vp',0.05});
  MOD = add_to_ee_model(MOD,'parm',{'Iv',3e-6,'Vv',0.3});
  MOD = add_to_ee_model(MOD,'parm',{'C', 1e-15});
  MOD = add_to_ee_model(MOD,'fe', @fe);
  MOD = add_to_ee_model(MOD,'qe', @qe);
  MOD = finish_ee_model(MOD);
end

function out = fe(S)
  v2struct(S);
  I_diode = Is*(exp(vpn/Vt)-1);
  I_tunnel = (Ip/Vp) * vpn * exp(-1/Vp * (vpn - Vp));
  I_excess = (Iv/Vv) * vpn * exp(vpn - Vv);
```

```
19    out = I_diode + I_tunnel + I_excess;
20  end
21
22  function out = qe(S)
23    v2struct(S);
24    out = C*vpn;
25  end
```

In Fig. 5.1, we have shown an excerpt from SPICE3's implementation of a regular diode model. Again, the whole model contains multiple files with every circuit analysis supported by SPICE3 coded inside. In contrast, the 25 lines of ModSpec code above describe the entire tunnel diode model; it works in every analysis in MAPP.

Although the tunnel diode example here is a simple two-terminal device, the ModSpec format supports more general devices, with multiple terminals, internal nodes, *etc.*, through its vector equations (5.5) and (5.6). ModSpec also supports specifying parameters, noise sources and other features (such as device-specific limiting and initialization) [55, 162]. Since the format itself makes modellers explicitly aware of important mathematical features of the model (such as the numbers of equations and unknowns involved, which equations are purely algebraic and which involve differential terms, *etc.*), many common modelling errors are eliminated.

Another implication of the differential equation format (5.5) and (5.6) is that ModSpec devices are not limited to any specific physical domain. Domain-specific attributes (*e.g.*, voltage/current concepts for electrical devices, together with related constraints such as KCL and KVL [55]) are layered on through a Network Interface Layer (NIL), an add-on structure within ModSpec. Specifying several NILs for a single ModSpec device makes it easy to model multi-physics devices, as illustrated in Fig. 5.6. Perhaps most importantly, writing a device model in ModSpec enables immediate, standalone testing and model debugging in MATLAB®, without necessarily relying on any of MAPP's analyses.

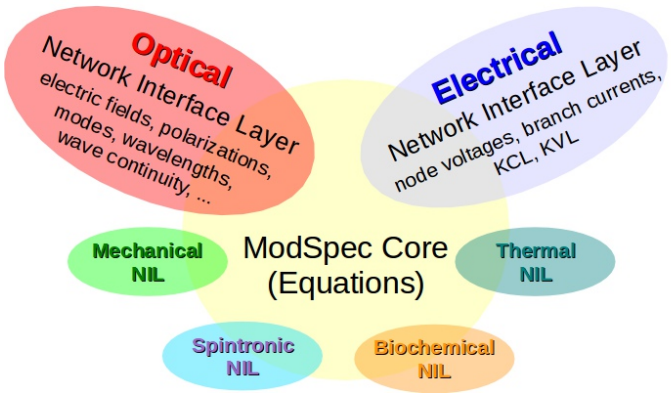



**Fig. 5.6:** ModSpec supports multiple physical domains in the same device through the concept of the Network Interface Layer (NIL).

Step 1.2: *Testing the model standalone*: The ModSpec format is a MATLAB® structure and contains executable function fields. It allows modellers to evaluate and visualize the model's functions right after it is coded in MAPP, without incorporation within a circuit. This is useful for checking equation correctness and catching simple bugs at an early stage of the model development flow. The functions tested at this point are the same ones called during circuit simulation; since no translation or interpretation is involved, model development and deployment more transparent and reliable. Existing Verilog-A based model development flows and tools do not provide a standalone check capability, since they necessarily involve translation/interpretation implemented in each simulator; the only way such a model can be exercised by the user is by writing test circuits and running them in the simulator.

Continuing with the tunnel diode example, by evaluating $\vec{f}_e$ with different input voltages $\vec{x} = vpn$, we can plot the I/V curve of the tunnel diode (shown in Fig. 5.7) without putting it in a circuit. The ModSpec model also contains automatically-generated functions for the derivatives of $\vec{q}_e$, $\vec{f}_e$, $\vec{q}_i$, $\vec{f}_i$, *etc.*; the derivatives are computed by MAPP's automatic differentiation package vecvalder [162] ("`help MAPPautodiff`" in MAPP). By evaluating $\partial\vec{f}_e/\partial\vec{x}$, we can calculate conductances and plot the G/V curve shown in Fig. 5.8.

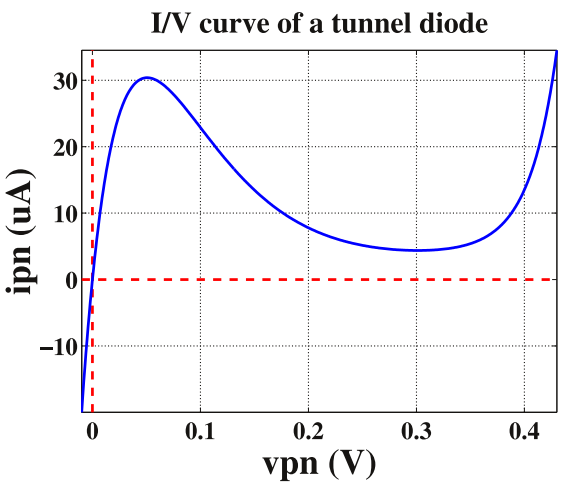



**Fig. 5.7:** I/V curve generated from the tunnel diode ModSpec model.

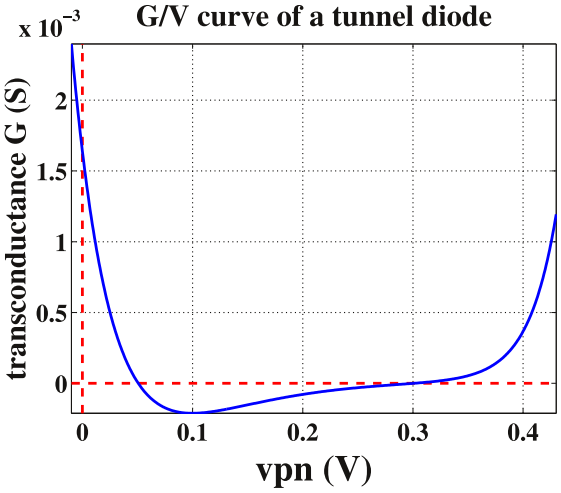



**Fig. 5.8:** G/V curve generated from the tunnel diode ModSpec model.

Furthermore, MAPP provides a Model Exerciser feature that allows users to plot curves like the ones in Fig. 5.7 and Fig. 5.8 with only a few lines of code. For example, we can initiate the model exerciser, then plot the I/V and G/V curves conveniently with the following MAPP code:

```
MEO = model_exerciser(Tunnel_Diode_ModSpec());
MEO.display(MEO); % displays available function names
          % and their usage, including 'ipn' and 'dipn_dvpn'
MEO.plot('ipn', 0:0.01:0.42, MEO);
MEO.plot('dipn_dvpn', 0:0.01:0.42, MEO);
```

Step 1.3: *Running the model within small circuits in MAPP*: Once the model has been examined standalone, it can be tested further in small circuits that are simulated in MAPP.

Frequently, this step reveals many problems (*e.g.*, bad numerics, unphysical results, connectivity issues, *etc.*) in newly-written models, especially those with equations that attempt to capture new physics. MAPP makes it easier to detect and address these problems than any other modelling/simulation framework we are aware of. MAPP's use of MATLAB® allows developers to debug their models effectively in an interactive coding environment. Also, MAPP's algorithm implementations are available to users as open source. They are object-oriented, mathematically-based, with functions inside clearly documented. They are meant to be easily accessible even by non-programmers. Thus debugging is more transparent to users. By running a new model within the DC, AC and transient algorithms within MAPP, most problems caused by the model for simulation can be detected.

For example, Fig. 5.9 shows a simple circuit where our previously introduced tunnel diode model is biassed within its negative resistance region by a voltage source, and connected with an RLC tank. With proper choice of parameters, the circuit becomes a negative-resistance LC oscillator. The circuit ("netlist") can itself be described in MAPP using MATLAB® commands ("`help MAPPcktnetlists`" for details). Running transient simulation in MAPP ("`help dot_transient`") demonstrates self-sustaining oscillation in the circuit, as illustrated in Fig. 5.10. From the standpoint of a device modeller, this provides important verification that the model can run in transient simulation. If transient fails, or if its results seem incorrect, the modeller gets to know immediately; the compact model's equations and/or their implementation can then be re-examined and corrected.

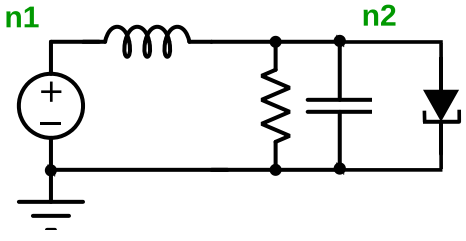



**Fig. 5.9:** Circuit schematic of an oscillator made with a tunnel diode.

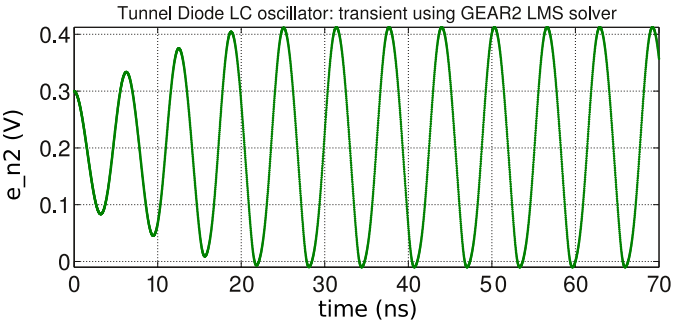



**Fig. 5.10:** Transient simulation results from the tunnel diode oscillator in Fig. 5.9.

Step 2: *Deploying the model*: Beyond providing facilities for testing models standalone and in small circuits, MAPP offers convenient and versatile tools to help prepare them for deployment. Most often, developers will wish to convert the ModSpec model into Verilog-A, the current industry standard for compact modelling [166], for public release. The fact that the model has already been tested and debugged on small circuits makes it far more likely that a Verilog-A version, if written properly, will work well in simulation.

The most likely cause for Verilog-A model problems at this step are discrepancies between the Verilog-A model and its ModSpec version, which are not hard to introduce, even for experienced compact modellers. To aid debugging, MAPP comes with a translator, VAPP (the Berkeley Verilog-A Parser and Processor), that can translate Verilog-A to ModSpec. Since ModSpec is an executable format, the translated ModSpec model can be conveniently compared against the original one, both by evaluating model functions and by running circuit simulations. If the two ModSpec models, one original, the other auto-translated from Verilog-A, are found to be consistent, considerable confidence is generated that the released Verilog-A model is correctly implemented. If not, debugging the auto-translated ModSpec model in the interactive environment of MAPP makes it easy, typically, to locate problems in the Verilog-A version.

Thus, not only does MAPP incorporate Verilog-A within its compact modelling flow, it also adds convenient visualization, testing and debugging features that can speed development and improve the quality of Verilog-A versions of a compact model.

In parallel with Verilog-A release, model developers can also directly release their ModSpec models in MATLAB®. Moreover, a C++ version of ModSpec is also available, using which models can be compiled standalone to generate dynamically-loadable libraries that conform to a C++ version of the ModSpec API.

Step 3: *Simulating the model*: Once the Verilog-A model is ready, it can be used by any simulator that supports compact model descriptions in Verilog-A. For example, the circuit in Fig. 5.9 was simulated in both Spectre and HSPICE. A snapshot of Spectre's results is shown in Fig. 5.11, while results from the HSPICE engine are plotted by MATLAB® in Fig. 5.12.

Furthermore, the C++ version of the ModSpec model can be simulated by any simulator that supports the C++ ModSpec API. We have implemented such support in the simulator Xyce [153] by writing a ModSpec interface within Xyce. This Xyce-ModSpec interface consists of less than 1000 lines of C++ code, and can dynamically link any C++ ModSpec model into Xyce. Fig. 5.12 overlays a Xyce simulation of the tunnel diode oscillator; a C++ ModSpec version of the tunnel diode model was incorporated into Xyce using the Xyce-ModSpec interface. We stress that results from the model, prototyped with MAPP, are identical in all simulators we have tried (Fig. 5.12 and Fig. 5.11), with Verilog-A and ModSpec deployments being consistent. The MAPP-based development flow we have outlined makes it far easier and faster to achieve such consistency than previous flows.

The ModSpec model format, being relatively new, is not widely supported in simulators yet. Nevertheless, its adoption can confer a number of advantages. Implementing a C++ ModSpec interface in a simulator is typically much easier than implementing Verilog-A support; once the former is done, any ModSpec model can be immediately used by the simulator by linking in its shared library dynamically. Supplying a model that conforms to the open and full-featured ModSpec API improves compatibility across different simulators; proprietary models can be deployed as pre-compiled dynamically-loadable binary libraries

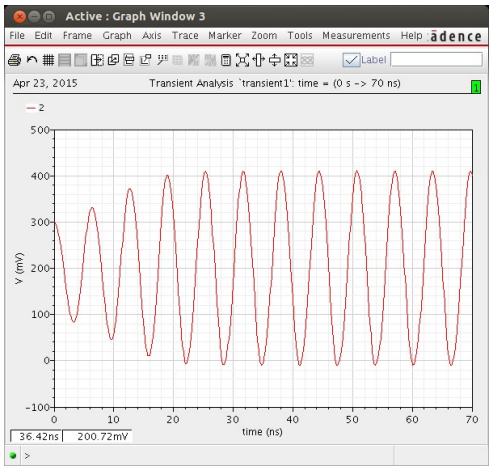



**Fig. 5.11:** Screenshot from Cadence® Virtuoso®, showing Spectre transient simulation results of the circuit in Fig. 5.9.

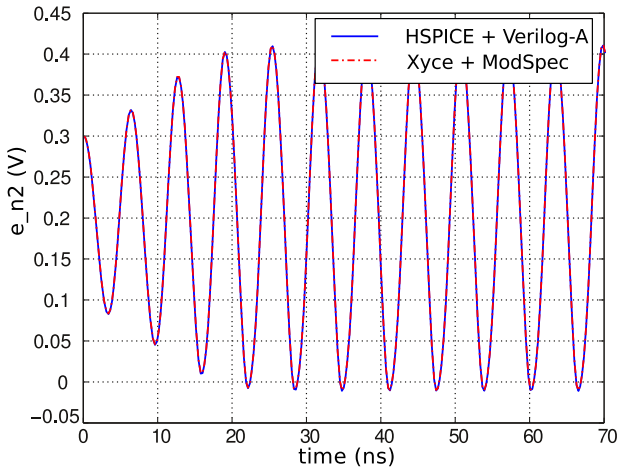



**Fig. 5.12:** Transient results from HSPICE and Xyce of the circuit in Fig. 5.9.

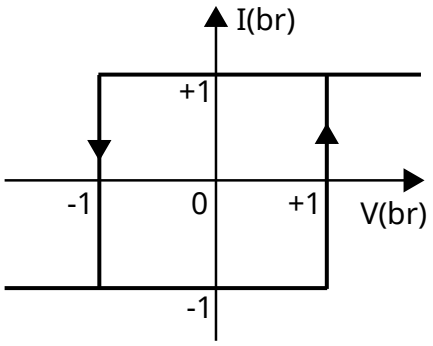
to help protect intellectual property (IP). Since ModSpec API functions can be called directly, without relying on any particular simulator or analysis, deployed models can tested standalone – another important features. These merits make ModSpec-based model deployment a useful complement to Verilog-A releases.

The ModSpec format and MAPP's model development flow are very useful in the study of device modelling. The remainder of this section provides two examples — its use in correctly modelling a type of nonlinear devices traditionally hard to model (Sec. 5.3.1), and MAPP's capability in modelling multiphysics devices/systems (Sec. 5.3.2).

### 5.3.1 Modelling Nonlinear Devices with Hysteresis

Many devices feature multiple stable equilibrium points. For example, in ESD protection devices, a phenomenon known as "snapback" in the $I$–$V$ characteristic curves allows the devices to draw different amounts of current at the same input voltage, depending on the occurrence of impact ionization [167]. Similarly, recently developed memristive devices can be in either high- or low-resistance state when powered off, depending on the voltages that have been applied to them before [168, 169]. In such devices with multistability, sweeping their inputs up and down often generates hysteresis, *i.e.*, a looping behavior in the input–output (I–O) graph.

These device properties are often difficult to model properly, or understand intuitively. In fact, compact model developers are often befuddled by them, resulting in a plethora of problematic models. Fig. 5.13 shows a very simple example; similar code with `if-else`




```
1 real i;
2 analog begin
3     if V(br) < -1
4         i = -1;
5     if V(br) > +1
6         i = +1;
7     I(br) <+ i;
8 end
```

**Fig. 5.13:** Example of problematic Verilog-A code for modelling *I–V* hysteresis.

statements and memory states[2] for modelling hysteresis can be found as part of several published compact models for ESD clamps [172] and memristors [173, 174].

We argue that the tools and platforms model developers use contribute substantially to this difficulty and confusion. Specifically, the way device models are implemented in SPICE-like simulators encourages developers to modify simulation algorithms inside devices for incorporating hysteresis. Similarly, although Verilog-A models are not written into the simulators, the language provides many confusing constructs developers can use to modify simulator behaviors [171, 175–177], enabling problematic code such as in Fig. 5.13. In comparison, MAPP and ModSpec enforce the use of the correct DAE format for device modelling; we show this correct way for modelling multistability and hysteresis in the remainder of this example section. But first, we would like to clarify several common confusions about these devices.

Firstly, although multistability normally implies that there will be a sudden jump in a device's response when sweeping its input, it does not mean there has to be discontinuity in the model equations. It doesn't justify the use of `if-else` statements either. In fact, continuous and smooth model equations can also create abrupt changes in device responses; designing such smooth equations is the key in modelling multistability and hysteresis properly.

Moreover, although hysteresis implies time dependence between inputs and outputs, it does not mean the device has to know the absolute time. Neither does it need to access the input at which it was evaluated at the last time point. In fact, a properly written compact model should not be specific to time-dependent simulation algorithms; it should run in other analyses, such as DC, small signal AC, Harmonic Balance, *etc.*, as well. Several existing compact models [173, 178] incorporate *I–V* hysteresis by accessing `$abstime` and implementing time integration inside; their use is limited to only transient simulation. There are also Verilog-A models that use memory states for storing and accessing the input

[2] A memory state, or hidden state [170], in Verilog-A is a variable used without assigning a value. They should be avoided in compact models [171].

value in the previous device evaluation, limiting their robustness in PSS simulations [170].

Another misconception is that a model needs to be an analog behavior model [179] to have hysteresis. This leads to the use of many simulator directives, *e.g.*, `@initial_step`, `analysis()`, `@cross()`, `$bound_step()`, *etc.*, whereas these constructs are in fact unnecessary for modelling hysteresis, and should be avoided [171, 177].

Multistability does not mean ill-posedness [180] — it does not mean there has to be a region in the state space with zero derivatives to keep the device output from moving. While some models [178, 181] use such "flat" regions for modelling multistability, this approach results in singular circuit Jacobian matrices and undefined model behaviors in these regions.

In fact, the basic requirements on a device model with multistability and hysteresis are no different from those on general compact models — the model should still be formulated in the DAE format; it should use continuous/smooth functions and should be well posed [95, 180, 182]. In Sec. 5.3.1.1, we show how this can be done in general by considering a simple two-terminal device with hystersis. We show that by including an internal state variable and designing its dynamics properly, *I–V* hysteresis can be included in the model. Specifically, we show how a key property of the model — a single continuous/smooth curve in the state space that contains all the steady state solutions, is connected to the well-posedness of the model, and how a negative-sloped fold in the curve results in the abrupt transitions observed in device responses. In the meanwhile, we also demonstrate the usefulness of the homotopy analysis [183], which was originally developed mainly to aid DC convergence, in characterizing and analyzing hysteretic devices. Then we write this example model both in the ModSpec format in MAPP, and in the Verilog-A language. For the Verilog-A implementation, we use the most consistently supported features of the language, such that the model will run in all main-stream simulators, including Spectre®, HSPICE and Xyce.

Then in Sec. 5.3.1.2 and Sec. 5.3.1.3, we apply the insights gained from studying the generic hysteretic model in Sec. 5.3.1.1 to more concrete device examples — ESD clamps and memristive devices.

We would like to note that the models developed and studied in this section all consist of relatively simple equations; the purpose is to illustrate the modelling methodology for devices with multistability and hysteresis. More complex versions of them, with more physical effects taken into consideration, are part of the future work.

#### 5.3.1.1 How to Model Hysteresis Properly

The equation of a general two-terminal device without memory can be written as

$$I(t) = f(V(t)), \tag{5.8}$$

where $V(t)$ is the voltage across the device, $I(t)$ the current through it. For example, $f(V(t)) = \frac{V(t)}{R}$ describes a simple linear resistor.

For devices with *I–V* hysteresis, $I(t)$ and $V(t)$ cannot have a simple algebraic mapping like (5.8). Instead, we introduce a state variable $s(t)$ into (5.8) and rewrite the *I–V* relationship as

$$I(t) = f_1(V(t),\ s(t)). \tag{5.9}$$

The dynamics of the internal state variable $s(t)$ is governed by a differential equation:

$$\frac{d}{dt}s(t) = f_2(V(t),\ s(t)). \tag{5.10}$$

In this formulation, we cannot directly calculate the current based on the voltage applied to the device at a single time $t$; $I(t)$ also depends on the value of $s(t)$. On the other hand, at time $t$, the value of $s(t)$ is determined by the history of $V(t)$ according to (5.10). Therefore, we can think of the device as having internal "memory" of the history of its input. If we choose the formula for $f_1$ and $f_2$ in (5.9) and (5.10) properly, as we sweep the input voltage, hysteresis in the current becomes possible.

In the rest of this section, (5.9) and (5.10) serve as a model template for devices with *I–V* hysteresis. To illustrate its use, we design a device example, namely "`hys_example`", with functions $f_1$ and $f_2$ defined as follows.

$$f_1(V(t),\ s(t)) = \frac{V(t)}{R} \cdot 0.5 \cdot (s(t)+1). \tag{5.11}$$

$$f_2(V(t),\ s(t)) = \frac{1}{\tau}\left(\tanh(K \cdot (V(t)+2 \cdot s(t))) - s(t)\right). \tag{5.12}$$

The choice of $f_1$ is easy to understand. If we assume $s(t)$ is within $(-1,1)$, incorporating $0.5 \cdot (s(t)+1)$ as a factor modulates the conductance of the device between 0 and $1/R$. The choice of $f_2$ determines the dynamics of $s(t)$. And when $f_2 = 0$, the corresponding $(V,\ s)$ pairs will show up as part of the DC solutions of circuits containing this device. Therefore, if we visualize the values of $f_2$ in a contour plot, such as in Fig. 5.14 (a), the curve representing $f_2 = 0$ is especially important. In (5.12), through the use of the tanh function plus a linear term in $s$, we design the $f_2 = 0$ curve to fold back in the middle, crossing the $V = 0$ axis three times. In this way, when $V$ is around 0, there are three possible values $s$ can settle on, all satisfying $\frac{d}{dt}s(t) = f_2 = 0$. This multiple stability in state variable $s$ is the foundation of hysteresis found in the DC sweeps on the device.

Fig. 5.14 (b) illustrates how hysteresis takes place in DC sweeps. In Fig. 5.14 (b), we divide the $f_2 = 0$ curve into three parts: curve `A` and `B` have positive slopes while `C` has a negative one. When we sweep $V$ towards the right at a very slow speed to approximate DC conditions, starting from a negative value left of $V-$, at the beginning, there is only one possible DC solution of $s$. As we increase $V$, the $(V,\ s)$ pair will move along curve `A`, until `A` ends when $V$ reaches $V+$. If $V$ increases slightly beyond $V+$, multiple stability in $s$ disappears. $(V,\ s)$ reaches the $f_2 > 0$ region and $s$ will grow until it reaches the `B` part of

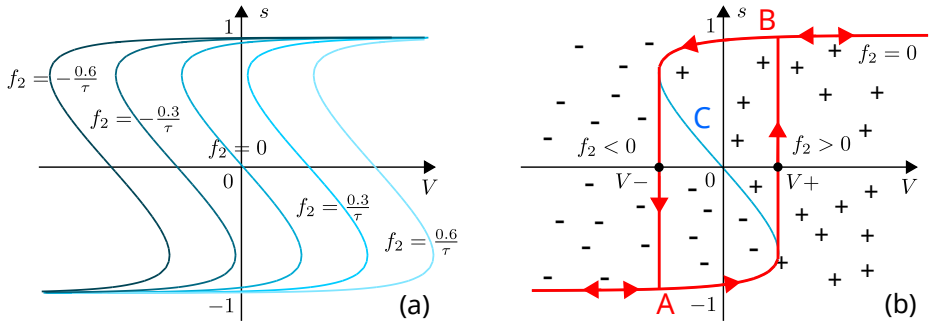



**Fig. 5.14:** Contour plot of $f_2$ function in (5.12) and predicted $s$-$V$ hysteresis curve based on the sign of $f_2$.

the $f_2 = 0$ curve. This shows up in the DC solutions as a sudden jump of $s$ towards curve B. Similarly, when we sweep $V$ in the other direction starting from the right of $V+$, the $(V, s)$ pair will follow curve B, then have a sudden shift to A at $V-$. Because $V+ > V-$, hysteresis occurs in $s$ when sweeping $V$, as illustrated in Fig. 5.14 (b). Since $s$ modulates the device's conductance, there will also be hysteresis in the $I$–$V$ relationship.

Note that the analysis of the origin of hysteresis does not involve absolute time. It is the fold in the DC solution curve defined by $f_2(v,\ s) = 0$ that generates multiple stable equilibriums in the device, which then result in abrupt changes and hysteresis when sweeping the device's input. As mentioned earlier, $s$ can be thought of as encoding the memory of $V$ from the past. Its multiple equilibriums reflect the different possible sets of history of $V$. And the separation between $V+$ and $V-$ in the DC curves ensures that no matter at what speed we sweep $V$, there will always be hysteresis in the $s$–$V$ relationship.

Model equations for `hys_example` defined in (5.11) and (5.12) can be written as a compact model into the MAPP using the ModSpec format. For `hys_example`,

$$\begin{aligned} \vec{f}_e(\vec{x},\vec{y},\vec{u}) &= \frac{\vec{x}}{R}\cdot(\tanh(\vec{y})+1), \quad \vec{q}_e(\vec{x},\vec{y}) = 0, \\ \vec{f}_i(\vec{x},\vec{y},\vec{u}) &= \tanh(K\cdot(\vec{x}+2\cdot\vec{y})) - \vec{y}, \quad \vec{q}_i(\vec{x},\vec{y}) = -\tau\cdot\vec{y}, \end{aligned} \tag{5.13}$$

with $\vec{x} = [V]$, $\vec{y} = [s]$, $\vec{z} = [I]$, $\vec{u} = []$.

```
function MOD = hys_ModSpec()
  MOD = ee_model();
  MOD = add_to_ee_model(MOD, 'name', 'hys');
  MOD = add_to_ee_model(MOD, 'terminals', {'p', 'n'});
  MOD = add_to_ee_model(MOD, 'explicit_outs', {'ipn'});
  MOD = add_to_ee_model(MOD, 'internal_unks', {'s'});
  MOD = add_to_ee_model(MOD, 'implicit_eqn_names', {'ds'});
  MOD = add_to_ee_model(MOD, 'parms', {'R', 1e3, ...
                                'K', 1, 'tau', 1e-3});
  MOD = add_to_ee_model(MOD, 'fqei', {@fe, @qe, @fi, @qi});
  MOD = finish_ee_model(MOD);
end % hys_ModSpec

function out = fe(S)
```

```
  v2struct(S); % populates workspace with vpn/R/K/tau
  out = vpn/R * 0.5 * (1+s); % ipn
end % fe

function out = qe(S)
  out = 0; % ipn
end % qe

function out = fi(S)
  v2struct(S);
  out = tanh(K*(vpn + 2*s)) - s;
end % fi

function out = qi(S)
  v2struct(S);
  out = - tau * s;
end % qi
```

**Listing 5.1:** `hys_example_ModSpec.m`

We can enter the model information in (5.13) into MAPP by constructing a ModSpec object `MOD`. The code in Listing 5.1 shows how to create this device model for `hys_example` entirely using MATLAB® in MAPP.

We can then simulate the model specified with Listing 5.1 using MAPP in various circuit analyses. Fig. 5.15 shows the results from DC sweep and transient simulation with input voltage sweeping up and down. It confirms that hysteresis takes place in both $I$–$V$ and $s$–$V$ relationships of the device.

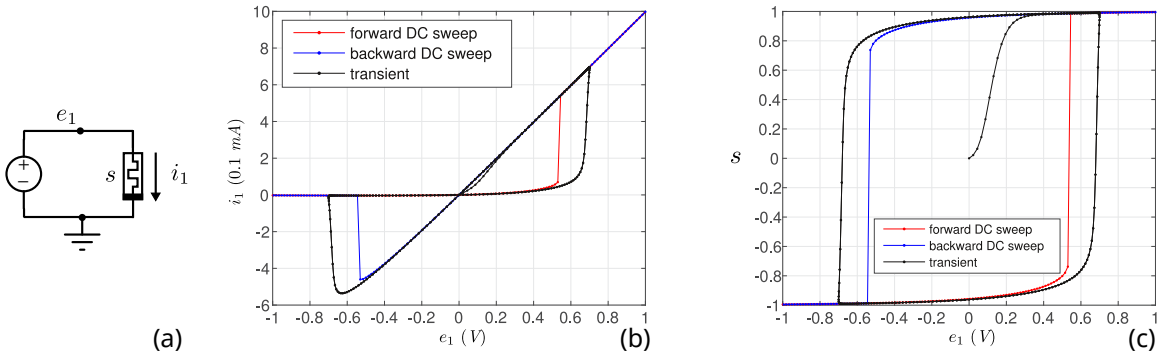



**Fig. 5.15:** Results from DC sweep and transient simulation in MAPP, showing hysteresis in both $s$ and $i_1$ when sweeping the input voltage, in either type of the analyses.

When we sweep $V$ back and forth, curve `C`, the one with a negative slope in Fig. 5.14 (b) never shows up in solutions. The reason is that, although it also consists of solutions of $f_2 = 0$, these solutions are not stable. With a little perturbation, whether from physical noise or numerical error, a $(V, s)$ point on curve `C` will move to either `A` or `B`. These unstable solutions can be captured using the homotopy analysis [183]. Homotopy analysis can track the DC solution curve in the state space. Results from homotopy analysis are shown in

Fig. 5.16. We note that all the circuit's DC solutions indeed form a smooth curve in the state space. The side view of the 3-D plot displays curve `C` we have designed in our model equation (5.12). The corresponding curve in the top view connects the two discontinuous DC sweep curves in Fig. 5.15; it consists of all the unstable solutions in the *I–V* relationship. These results from homotopy analysis provide us with important insights into the model. They reveal that there is a single smooth and continuous DC solution curve in the state space, which is an indicator of the well-posedness of the model. They also illustrate that it is the fold in the smooth DC solution curve that has created the discontinuities in DC sweep results. These insights are important for the proper modelling of hysteresis.

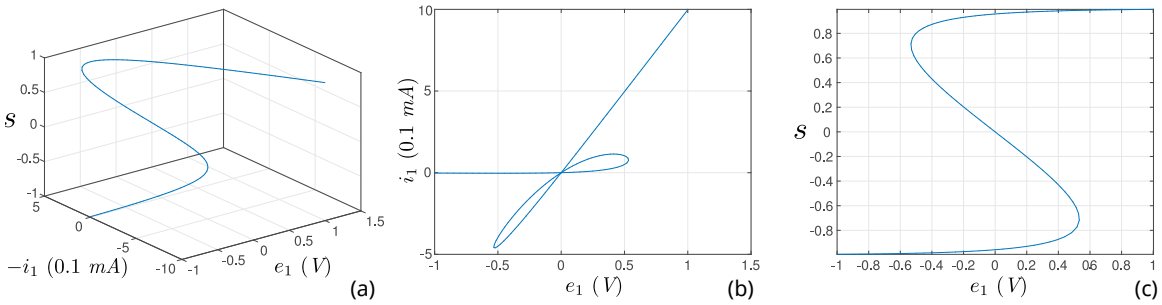



**Fig. 5.16:** Results from homotopy analysis in MAPP: (a) 3-D view of all the DC solutions; (b) top view of the DC solutions shows the folding in the *I–V* characteristic curve, explaining the *I–V* hysteresis from DC and transient voltage sweeps in Fig. 5.15; (c) side view of the DC solutions.

Moreover, the top view explains the use of internal state *s* for modelling hysteresis from another angle. Without the internal state, it would be difficult if not impossible to write a single equation describing the *I–V* relationship shown in Fig. 5.16 (b). With the help of *s*, we can easily choose two simple model equations as (5.11) and (5.12), and the complex *I–V* relationship forms naturally.

```
1  `include "disciplines.vams"
2  module hys(p, n);
3      inout p, n;
4      electrical p, n, ns;
5      parameter real R = 1e3 from (0:inf);
6      parameter real K = 1 from (0:inf);
7      parameter real tau = 1e-3 from (0:inf);
8      real s;
9
10     analog begin
11         s = V(ns, n);
12         I(p, n)  <+ V(p, n)/R * 0.5 * (1+s);
13         I(ns, n) <+ tanh(K*(V(p, n) + 2*s)) - s;
14         I(ns, n) <+ ddt(-tau*s);
15     end
16 endmodule
```

**Listing 5.2:** `hys_example.va`

`hys_example` can also be implemented in the Verilog-A language. Apart from the differences in syntax, Verilog-A differs from ModSpec in one key aspect — the way of handling internal unknowns and implicit equations. Verilog-A models a device with an internal circuit topology, *i.e.*, with internal nodes and branches defined just like in a subcircuit. The variables in a Verilog-A model, the "sources" and "probes", are potentials and flows specified based on this topology. Coming from this subcircuit perspective, the language doesn't provide a straightforward way of dealing with general internal unknowns and implicit equations inside the model, *e.g.*, the state variable $s$ and the equation (5.10) in `hys_example`.

As a result, an internal unknown is often declared as a general variable using the `real` statement. `idt()`, `$abstime` and hard-coded time integration methods are often used for describing implicit differential equations. These approaches should be avoided in modelling [95, 171]. Instead, in this section, we show how to properly model both the state variable $s$ by considering it as a voltage, and the implicit equation by treating it as the KCL at an internal node. As in Listing 5.2, we declare an internal branch, whose voltage represents $s$. One end of the branch is an internal node that doesn't connect to any other branches. In this way, by contributing $\tanh(K\cdot(V+2\cdot s))-s$ and `ddt(-tau * s)` both to this same branch, the KCL at the internal node will enforce the implicit differential equation in (5.12).

Declaring $s$ as a voltage is not the only way to model `hys_example` in Verilog-A. Depending on the physical nature of $s$, one can also use Verilog-A's multiphysics support and model it as a potential in other desciplines. One can also switch potential and flow by defining $s$ as a flow instead. The essence of our approach is to recognize that state variable $s$ is a circuit unknown, and thus should be modelled as a potential or flow in Verilog-A, for the consistent support from different simulators in various circuit analyses.

#### 5.3.1.2 Modelling ESD Snapback

ESD protection devices feature a phenomenon known as snapback — the current through a device does not monotonically grow with the input voltage, but folds back within a certain voltage range. This fold in the *I–V* graph can be observed in Transmission Line Pulse (TLP) measurements. It physically means that, when the device is put in a circuit, as its input voltage increases beyond a certain trigger point, namely $V_{t1}$, impact ionization begins to happen and the amount of current through the device suddenly jumps. And the high current can sustain itself when the voltage is swept back to $V_{t1}$; the device will turn "off" only at a lower voltage when the current drops below a hold current $I_H$, corresponding to a voltage $V_{I_H}$ normally smaller than $V_{t1}$. In between $V_{I_H}$ and $V_{t1}$, the device can have different currents depending on whether it is in the "on" or "off" state.

To incorporate such devices in circuit simulation, some specialized algorithms have been developed [184, 185]. As for compact models, some physics-based ones leverage existing models for BJTs and MOS devices and design subcircuits around them for approximating the device structure and characteristics [186–188]. In comparison, behavioral models for ESD

clamps [167, 189, 190] have much lower model complexity, which simplifies parameter extraction significantly and provides more intuitions into the operation of the devices. Among the available behavioral models, [167] is the first to be able to capture the time dependence in the on/off transition in ESD protection devices. The model discussed in this section is based on it.

From the discussion in Sec. 5.3.1.1, we note the similarity between the ESD snapback behavior and the model template we have developed for a general hysteretic device. This indicates that the multistability observed in ESD protection devices can also be modelled by introducing a state variable $s$, and designing a fold in the steady state curve of its dynamics. Adapted from [167], we model the $I$–$V$ relationship as

$$I = I_{off} + s \cdot I_{on}, \tag{5.14}$$

where $I_{on}$ and $I_{off}$ are empirical equations for on-state and off-state currents:

$$I_{on} = G_{on} \cdot (V - V_H), \tag{5.15}$$

$$I_{off} = I_S \cdot e^{-V/V_T} \cdot \sqrt{1 + \frac{\max(V,0)}{V_D}}. \tag{5.16}$$

Here, $s$ is a state variable between 0 and 1; it is an indicator of whether impact ionization is present. It should grow to 1 when $V > V_{t1}$, and fall back to 0 when $V < V_{I_H}$; in between, it can have multiple steady state values.

The dynamics of this state variable can be modelled in similar ways as discussed in Sec. 5.3.1.1. In the formulation of the growth of $s$ in (5.12), the steady state of $s$ is naturally limited to $(-1,\ 1)$; we convert it to $(0,\ 1)$ by using $s^* = 2 \cdot (s - 0.5)$ in (5.12) instead. Similarly, in (5.12), when the voltage across the device is swept up and down, the transition voltage thresholds are around $\pm 1$; we bring these thresholds to $V_{t1}$ and $V_{I_H}$ by first convert $V$ to $V^*$ before putting it in (5.12).

$$V^* = \frac{2}{V_{t1} - V_{I_H}} \cdot (V - 0.5 \cdot V_{t1} - 0.5 \cdot V_{I_H}). \tag{5.17}$$

Then the dynamic of the internal ionization indicator state is modelled as follows.

$$\tau \cdot \frac{d}{dt} s = \tanh(K \cdot (V^* + 2 \cdot s^*)) - s^*. \tag{5.18}$$

```
function MOD = ESD_snapback_ModSpec()
  MOD = ee_model();
  MOD = add_to_ee_model(MOD, 'name', 'ESD snapback');
  MOD = add_to_ee_model(MOD, 'terminals', {'p', 'n'});
  MOD = add_to_ee_model(MOD, 'explicit_outs', {'ipn'});
  MOD = add_to_ee_model(MOD, 'internal_unks', {'s'});
  MOD = add_to_ee_model(MOD, 'implicit_eqn_names', {'ds'});
```

```
  MOD = add_to_ee_model(MOD, 'parms', {'Gon', 0.01,...
      'VH', 16, 'VT1', 48, 'VIH', 26, 'Is', 1e-12,...
      'VT', 0.026, 'VD', 0.7, 'K', 1, 'tau', 1e-9,...
      'C', 1e-13, 'maxslope', 1e15, 'smoothing', 1e-10});
  MOD = add_to_ee_model(MOD, 'fqei', {@fe, @qe, @fi, @qi});
  MOD = finish_ee_model(MOD);
end

function out = fe(S)
  v2struct(S);
  Ion = smoothclip(Gon*(vpn - VH), smoothing)...
        - smoothclip(-Gon*VH, smoothing);
  Ioff = Is * (1 - safeexp(-vpn/VT, maxslope))...
         * sqrt(1 + max(vpn, 0)/VD);
  out = Ioff + s * Ion; % ipn
end

function out = qe(S)
  v2struct(S);
  out = C * vpn;
end

function out = fi(S)
  v2struct(S);
  Vstar = 2*(vpn-0.5*VT1-0.5*VIH)/(VT1-VIH);
  sstar = 2*(s-0.5);
  out = tanh(K*(Vstar + 2*sstar)) - sstar;
end

function out = qi(S)
  v2struct(S);
  out = -tau*s;
end
```

**Listing 5.3:** `ESD_snapback_ModSpec.m`

```
`include "disciplines.vams"
module ESDsnapback(p, n);
  inout p, n;
  electrical p, n, ns;

  parameter real Gon = 0.1 from (0:inf);
  parameter real VH = 16 from (0:inf);
  parameter real VT1 = 48 from (0:inf);
  parameter real VIH = 26 from (0:inf);
  parameter real Is = 1e-12 from (0:inf);
  parameter real VT = 0.026 from (0:inf);
  parameter real VD = 0.7 from (0:inf);
  parameter real K = 1 from (0:inf);
  parameter real C = 1e-13 from [0:inf);
  parameter real tau = 1e-9 from (0:inf);
  parameter real maxslope = 1e15 from (0:inf);
  parameter real smoothing = 1e-10 from (0:inf);
  real s, Ion, Ioff, Vstar, sstar;

  analog function real smoothclip;
    input x, smoothing;
    real  x, smoothing;
    begin
      smoothclip = 0.5*(sqrt(x*x + smoothing) + x);
    end
```

```
26    endfunction // smoothclip
27
28    analog begin
29      s = V(ns, n);
30      Ion = smoothclip(Gon*(V(p, n)-VH), smoothing)
31            - smoothclip(-Gon*VH, smoothing);
32      Ioff = Is * (1 - limexp(-V(p, n)/VT))
33             * sqrt(1 + max(V(p, n), 0)/VD);
34      Vstar = 2*(V(p, n)-0.5*VT1-0.5*VIH)/(VT1-VIH);
35      sstar = 2*(s-0.5);
36      I(p, n) <+ Ioff + s * Ion;
37      I(p, n) <+ ddt(C * V(p, n));
38      I(ns, n) <+ tanh(K*(Vstar + 2*sstar)) - sstar;
39      I(ns, n) <+ ddt(-tau*s);
40    end
41  endmodule
```

**Listing 5.4:** `ESD_snapback.va`

This is essentially the same $f_2$ function in the model template from Sec. 5.3.1.1, with its steady state solutions forming a similar curve as in Fig. 5.14 with a similar negative-sloped fold in the middle. The fold explains the multiple stable currents within $(V_{I_H}, V_{t1})$, as well as the $I$–$V$ hysteresis in the device. Equations (5.14) and (5.18) then constitute a behavioral model for ESD protection devices.

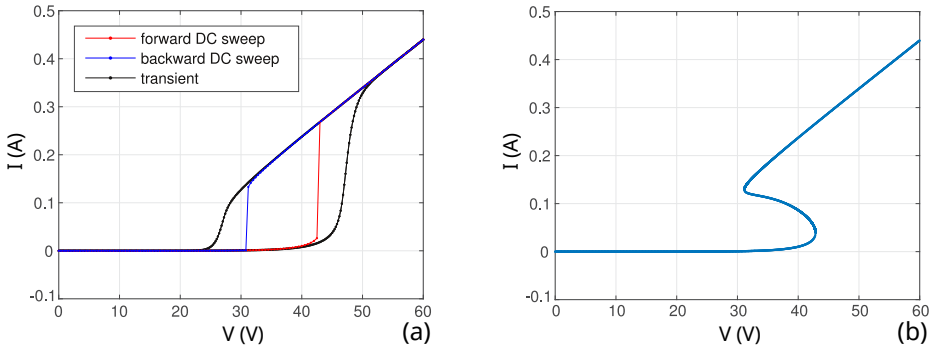



**Fig. 5.17:** Forward/backward DC, transient voltage sweep responses, and homotopy analysis results from the ESD clamp model in Listing 5.3.

The code implementation in ModSpec and Verilog-A are shown in Listing 5.3 and Listing 5.4 respectively. Fig. 5.17 shows simulation results from MAPP. Results from DC and transient voltage sweeps are overlaid, demonstrating the hysteresis in the $I$–$V$ graph; homotopy results are plotted in Fig. 5.17 (b), confirming the fold we have designed in the model's steady state curve. Transient results in Fig. 5.17 (a) also show that ionization does not happen instantaneously; same as [167], our model captures the time dependence of impact ionization. Moreover, on top of [167], our model also captures the $I$–$V$ hysteresis in DC sweeps. And it does so without sacrificing the model's smoothness or its robustness in simulation. The model works well in various circuits. As an illustration, we simulate an ESD clamp with the HBM configuration in Fig. 5.18. Transient results confirm that it

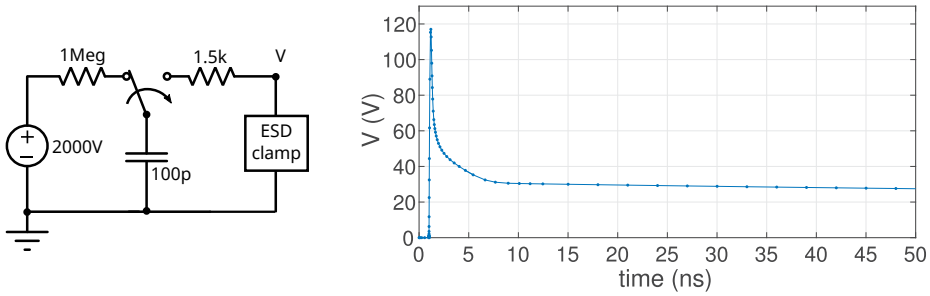



**Fig. 5.18:** HBM test bench for an ESD clamp and transient simulation results for the voltage across the clamp.

implements a clamp at $V_{I_H} \approx 30V$. Note that there is certain arbitrariness in the choice of the equation for the dynamics of the internal state; we are simply reusing the equation in (5.12) to illustrate the idea of modelling ESD snapback with a fold in the model's steady state solution curve. The transition points in (5.12) are not exactly $\pm 1$, and the exact value of $d/dt\ s$ is mainly controlled by the order of the time constant parameter $\tau$. This choice is partly due to the fact that there are currently no well-established formulae for the growth rate of impact ionization. Making the ionization dynamics more physical is part of the future research in the development of simulation-ready ESD clamp models.

#### 5.3.1.3 RRAM and Memristor Models

Existing models for memristive devices (RRAMs and other memristors) all suffer from issues related to mathematical ill-posedness. In particular, as noted in [95], they don't generate correct DC responses. Here, we show that the reason for the DC failure is indeed the lack of a single DC solution curve in steady state. Guided by the model template, we propose well-posed models for memristors. They preserve the accuracy and physics in existing models, while fixing their model problems. The result is a collection of models for various types of memristors, all working in all the common circuit analyses in major simulators.

An RRAM consists of two metal electrodes and a thin oxide film separating them. Depending on whether a conductive filament in the film connects the electrodes, the device can be in either low- or high-resistance state. Therefore, the internal state variable for RRAM models can be defined as the *gap* between the tip of the filament and the opposing electrode. By filling in the model template in Sec. 5.3.1.1 and designing the $f_1$ equation for current calculation and $f_2$ for *gap* dynamics, we will have compact models for RRAM devices.

Among the existing models for RRAMs and other memristive devices, the formula for $f_1$ are mostly consistent [21, 173, 174, 181]. In this section, we choose to use the $f_1$ function

in [173, 174]:

$$f_1(V,\ gap) = I_0 \cdot \exp(-\frac{gap}{g_0}) \cdot \sinh(\frac{V}{V_0}), \tag{5.19}$$

where $I_0$, $g_0$, $V_0$ are fitting parameters.

For $f_2$, we can adapt the *gap* growth formulation in [173, 174] and write it as

$$f_2(V,\ gap) = -v_0 \cdot \exp(-\frac{E_a}{V_T}) \cdot \sinh(\frac{V \cdot \gamma \cdot a_0}{t_{ox} \cdot V_T}), \tag{5.20}$$

where $v_0$, $E_a$, $a_0$ are fitting parameters, $t_{ox}$ is the thickness of the oxide film, $V_T = k \cdot T/q$ is the thermal voltage, and $\gamma$ is the local field enhancement factor [191].

While there are small differences among the $f_2$ functions in models developed by various groups [21, 173, 174, 181], they differ mainly in the definitions of fitting parameters. A property they all share is that the sign of $f_2$ is the same as that of $(-\sinh(vtb))$. Put in other words, *gap* begins to decrease whenever *vtb* is positive, and vice versa, as illustrated in Fig. 5.19 (a). While there is some physical truth to this statement, considering that an RRAM device will eventually be destroyed if applied a constant voltage for an indefinite amount of time, for the model to work in numerical simulation, the state variable *gap* has to be bounded.

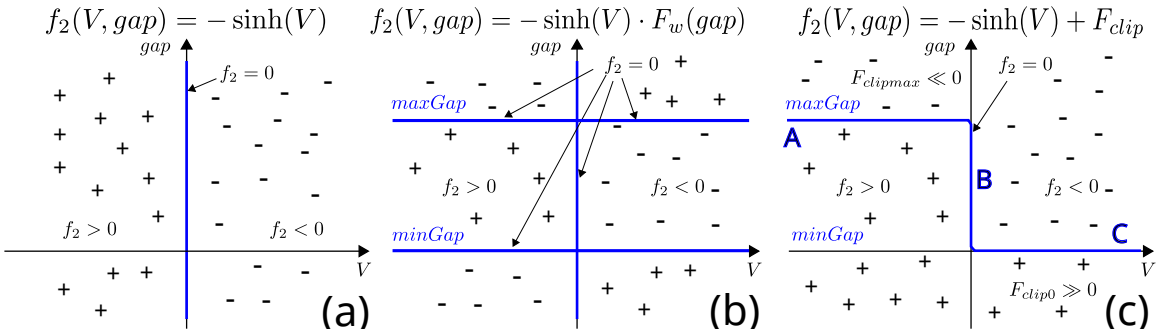



**Fig. 5.19:** Illustration of several choices of $f_2$ in RRAM model.

Ensuring that the upper and lower bounds for *gap* are always respected in simulation is one major challenge for the compact modelling of RRAM devices. To address this challenge, several approaches have been attempted in the existing RRAM compact models. Some models [173, 174] directly use `if-then-else` statements on *gap*. They declare `gap` as a `real` variable in Verilog-A, then directly enforce "`if (gap < 0) gap = 0;`". This practice excludes the model from the differential equation framework; they are not suitable for simulation analyses. Another category of models multiply the $f_2$ in (5.20) with a window function [21, 178, 181] that sets $\frac{d}{dt}gap = f_2 = 0$ when $gap = maxGap$ and $gap = minGap$. Such a window function is constructed either by directly using `step()` functions [21], or by adapting from some smooth windows, such as Joglekar [192], Biolek and Prodromakis

windows [193]. However, there are subtle and deeper problems with this approach. The problems can be illustrated by analyzing the sign and zero-crossings of function $f_2$ shown in Fig. 5.19 (b). The $f_2 = 0$ curves consist of three lines: the *maxGap* and *minGap* lines, and the $V = 0$ line, with two intersections. Some parts of these lines, such as the ones for ($gap > maxGap$), ($V > 0,\ gap \approx maxGap$) and ($V < 0,\ gap \approx minGap$), are model artifacts rather than physical steady state solutions. The existence of multiple DC curves can result in unphysical results in DC and transient simulations, and also cause convergence issues for homotopy analysis [95].

In our model, we try to bound variable *gap* while keeping the DC solutions in a single continuous curve, illustrated as the $f_2 = 0$ curve in Fig. 5.19 (c). This is inspired by studying the model template `hys_example` in Sec. 5.3.1.1. The sign and zero-crossing of $f_2$ for our RRAM model are closely related to those of the $f_2$ function for `hys_example`. To construct the desired $f_2 = 0$ solution curve, we modify the original $f_2$ in (5.20) by adding clipping terms to it that are smooth and continuous. The clipping terms can also leave the values from the original $f_2$ function in (5.20) almost intact when $minGap < gap < maxGap$. The code implementations in ModSpec and Verilog-A are shown in Listing 5.5 and Listing 5.6 respectively. Simulations in various simulators confirm that the models work robustly; some transient simulation results from MAPP are provided in Fig. 5.20.

```
function MOD = RRAM_ModSpec()
  MOD = ee_model();
  MOD = add_to_ee_model(MOD, 'name', 'RRAM');
  MOD = add_to_ee_model(MOD, 'terminals', {'t', 'b'});
  MOD = add_to_ee_model(MOD, 'explicit_outs', {'itb'});
  MOD = add_to_ee_model(MOD, 'internal_unks', {'Gap'});
  MOD = add_to_ee_model(MOD, 'implicit_eqn_names',...
                                              {'dGap'});
  MOD = add_to_ee_model(MOD, 'parms', {'g0', 0.25,...
        'V0', 0.25, 'I0', 1e-3, 'Vel0', 10,...
        'Beta', 0.8, 'gamma0', 16, 'Ea', 0.6,...
        'a0', 0.25, 'tox', 12});
  MOD = add_to_ee_model(MOD, 'parms', {'maxGap', 1.7,...
        'minGap', 0, 'maxslope', 1e15,...
        'smoothing', 1e-8, 'Kclip', 50, 'GMIN', 1e-12});
  MOD = add_to_ee_model(MOD, 'fqei', {@fe,@qe,@fi,@qi});
  MOD = finish_ee_model(MOD);
end

function out = fe(S)
  v2struct(S);
  out = I0*safeexp(-Gap/g0, maxslope) ...
        * safesinh(vtb/V0, maxslope) + GMIN*vtb;
end

function out = qe(S)
  out = 0; % itb
end

function out = fi(S)
  v2struct(S);
  T = 300;
  k = 1.3806226e-23; % Boltzmann's Constant
  q = 1.6021918e-19; % Electron Charge
```

```
35   Gamma = gamma0 - Beta * Gap^3;
36   ddt_Gap = - Vel0 * exp(- q*Ea/k/T) ...
37             * safesinh(vtb*Gamma*a0/tox*q/k/T, maxslope);
38   Fw1 = smoothstep(minGap-Gap, smoothing);
39   Fw2 = smoothstep(Gap-maxGap, smoothing);
40   clip_minGap = (safeexp(Kclip*(minGap-Gap), maxslope) ...
41                  - ddt_Gap) * Fw1;
42   clip_maxGap = (-safeexp(Kclip*(Gap-maxGap), maxslope) ...
43                  - ddt_Gap) * Fw2;
44   out = ddt_Gap + clip_minGap + clip_maxGap;
45 end
46
47 function out = qi(S)
48   v2struct(S);
49   out = - 1e-9 * Gap;
50 end
```

**Listing 5.5:** `RRAM_ModSpec.m`

```
1  `include "disciplines.vams"
2  `include "constants.vams"
3  module RRAM(t, b);
4    inout t, b;
5    electrical t, b, nGap;
6    parameter real g0 = 0.25 from (0:inf);
7    parameter real V0 = 0.25 from (0:inf);
8    parameter real Vel0 = 10 from (0:inf);
9    parameter real I0 = 1e-3 from (0:inf);
10   parameter real Beta = 0.8 from (0:inf);
11   parameter real gamma0 = 16 from (0:inf);
12   parameter real Ea = 0.6 from (0:inf);
13   parameter real a0 = 0.25 from (0:inf);
14   parameter real tox = 12 from (0:inf);
15
16   parameter real maxGap = 1.7 from (0:inf);
17   parameter real minGap = 0.0 from (0:inf);
18
19   parameter real smoothing = 1e-8 from (0:inf);
20   parameter real GMIN = 1e-12 from (0:inf);
21   parameter real Kclip = 50 from (0:inf);
22
23   real Gap, ddt_gap, Gamma, Fw1, Fw2;
24   real clip_minGap, clip_maxGap;
25
26   analog function real smoothstep;
27     input x, smoothing;
28     real  x, smoothing;
29     begin
30       smoothstep = 0.5*(x/sqrt(x*x + smoothing)+1);
31     end
32   endfunction // smoothstep
33
34   analog begin
35     Gap = V(nGap, b);
36     I(t, b) <+ I0 * limexp(-Gap/g0) * sinh(V(t, b)/V0)
37            + GMIN*V(t, b);
38
39     Gamma = gamma0 - Beta * pow(Gap, 3);
40     ddt_gap = -Vel0 * exp(-Ea/$vt) * sinh(V(t, b)
41               * Gamma*a0/tox/$vt);
42
```

```
43      Fw1 = smoothstep(minGap-Gap, smoothing);
44      Fw2 = smoothstep(Gap-maxGap, smoothing);
45      clip_minGap = (limexp(Kclip*(minGap-Gap))
46                     - ddt_gap) * Fw1;
47      clip_maxGap = (-limexp(Kclip*(Gap-maxGap))
48                     - ddt_gap) * Fw2;
49
50      I(nGap, b) <+ ddt_gap + clip_minGap + clip_maxGap;
51      I(nGap, b) <+ ddt(-1e-9*Gap);
52    end
53  endmodule
```

**Listing 5.6:** `RRAM.va`

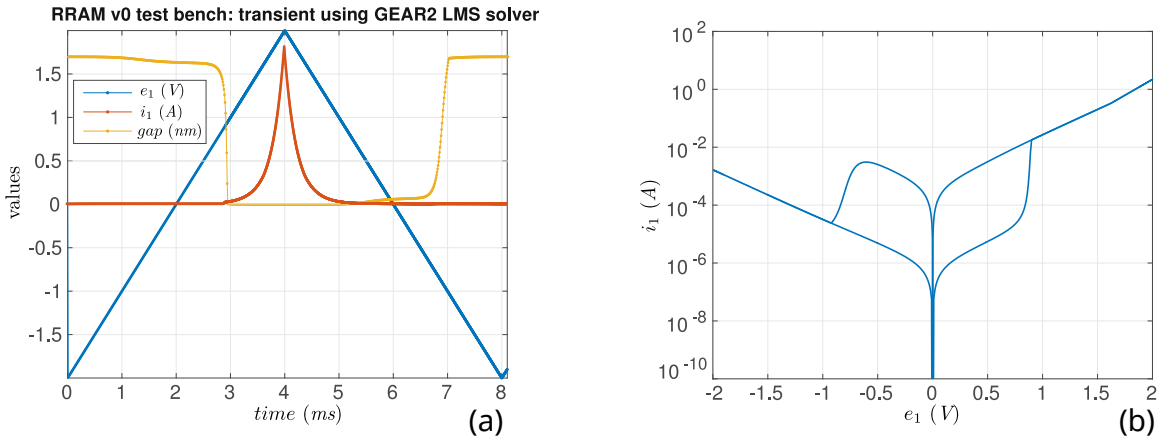



**Fig. 5.20:** Transient results on the circuit (same as in Fig. 5.15) with a voltage source connected to an RRAM device.

While the intention of adding the clipping terms is to set up bounds for variable *gap* and to construct DC solution curve in Fig. 5.19 (c), there is also some physical justification to our approach. As a physical quantity, *gap* is indeed bounded by definition. Therefore, $\frac{d}{dt}gap = f_2$ cannot look like Fig. 5.19 (a) or (b) in reality. The $f_2 = 0$ curves must have the `A` and `B` parts labelled in Fig. 5.19 (c). One can think of the clipping terms as infinite amount of resisting "force" to keep *gap* from decreasing below *minGap*, or increasing beyond *maxGap*. The analogy is the modelling of MEMS switches, where the switching beam's position is often used as an internal state variable. This variable reaches its bound when the switching beam hits the opposing electrode (often the substrate). The position does not move further. The beam cannot move into the electrode because of the huge force resisting it from causing any shape change in the structures. Similarly, in RRAM modelling, if the variable *gap* represents it physical meaning accurately, one can expect such "forces" to exist to make it a bounded quantity. This physics intuition matches well with our proposed numerical technique of using fast growing exponential components to enforce the bounds.

Note that the DC curve for the RRAM model in Fig. 5.19 (c) does not bend with an opposite slope in the middle; the model is at the "cusp" of hysteresis and multistability. This

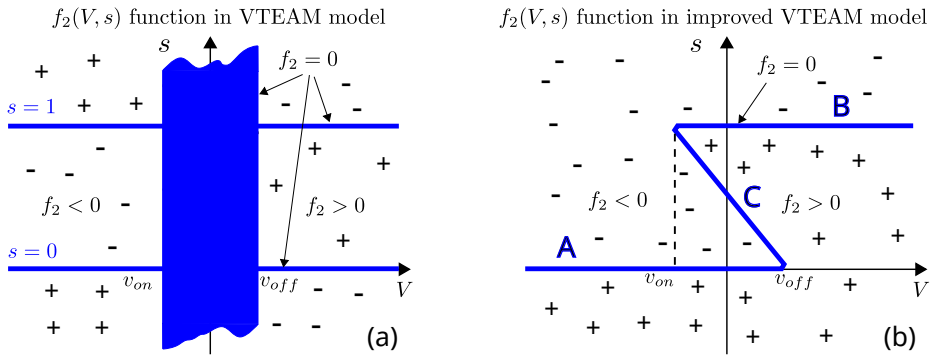



**Fig. 5.21:** $f_2$ function in VTEAM memristor model contains a flat region around $V = 0$ for the modelling of DC hysteresis. The proper way is to design a single solution curve of $f_2 = 0$ that folds back around $V = 0$.

matches physical reality, since RRAMs are considered non-volatile memories only within a certain data retention time. Apart from RRAMs, several compact models are designed for other memristive devices, including those with true DC hysteresis. As mentioned above as one of the confusions developers have for modelling hysteresis, they [178, 181] use regions with flat $f_2$ functions for modelling the multistability, resulting in the zeros-crossings of $f_2$ forming an area shown in Fig. 5.21 (a). From the discussion in this section, to model the same effect in these memristive devices while respecting the well-posedness of the model, the steady state curve should resemble the one shown in Fig. 5.21 (b) instead. Once we have this understanding, it becomes easy to design the $f_2$ functions in more memristor models [95] to bring about the desired DC solution curve as in Fig. 5.21 (b).

Thus, we have shown how the ModSpec format and MAPP enable us to come up with the proper way of modelling multistability and hysteresis. The results are not just simulation-ready well-posed models for ESD clamps and memristive devices, but also the general modelling methodology that applies to a wide class of nonlinear devices.

### 5.3.2 Modelling Multiphysics Devices and Systems

MAPP is designed from the ground up to support multiphysics modelling and simulation. Developers can write compact models for not only electronic devices, but also those from optoelectronics, spintronics, microelectromechanical systems (MEMS), *etc.*. They can also connect these device models into systems using multiphysics netlists and run various simulation algorithms on them. In this section, we explain the key concepts and techniques behind these modules, and illustrate the usage of them through examples.

The key enabling feature is, again, the use of DAE for modelling devices and systems. DAEs are mathematical concepts describing the relations between unknowns using equations and are thus not limited to electronics. As described in Sec. 5.2, ModSpec devices are DAEs; device connectivity is specified using a flexible MATLAB®-structure-based netlist; an equation engine processes both of them to construct the system DAEs. As a result,

MNA is implemented as only one of the equation engines. Sparse Tableu [55] analysis can be applied by swapping just the equation engine part while keeping devices and netlists unchanged [143].

From above, the support for multiphysics in MAPP consists of three parts: ModSpec models for multiphysics devices, circuit/system netlist with multi-domain connections, and an equation engine to process them.

At the device level, a ModSpec model includes an array of NILs for encoding I/O information from multiple physical domains within a single device model.

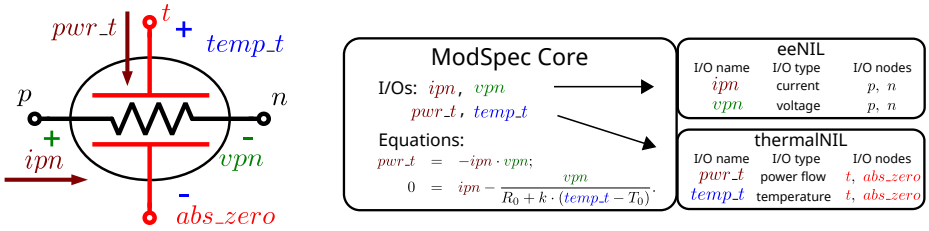



**Fig. 5.22:** Thermistor model with both electrical and thermal nodes. eeNIL and thermalNIL associate the I/Os in the ModSpec equations with their physical meanings.

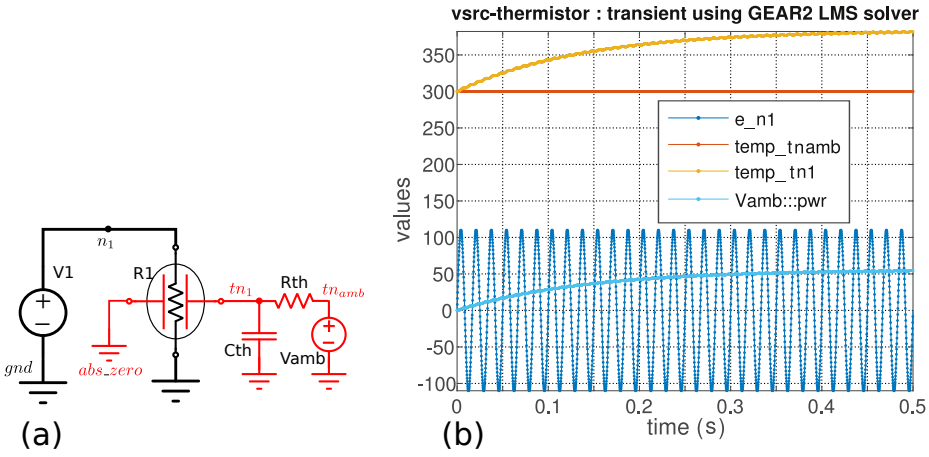



**Fig. 5.23:** Multiphysics circuit with a thermistor and its transient simulation results

As an example, Fig. 5.22 illustrates a thermistor model, where the core model specifies equations using variables *vpn*, *ipn*, *temp_t* and *pwr_t*. Two NILs, `eeNIL` and `thermalNIL`, are attached to this model to inform the equation engines of the physical meanings of these variables — branch voltage, current, temperature and power flow.

Such a model can be connected to a system such as the one in Fig. 5.23 by specifying a multiphysics netlist, *e.g.*, Fig. 5.24, completely using the MATLAB®language.

To support these multiphysics devices and netlists in MAPP, the key technique, as illustrated in Fig. 5.25, is to separate the components that are dependent on physical domains

```
function netlist = vsrc_thermistor_netlist()
  netlist.name = 'vsrc-thermistor';

  % regular voltage source V1
  netlist = add_element(netlist, vsource(), 'V1', {'electrical', '1', 'gnd'}, {}, {'dc', 1});
  % thermistor R1
  netlist = add_element(netlist, thermistor(), 'R1', {{'electrical', '1', 'gnd'}, ...
                                                      {'thermal', 't', '0'}});
  % thermal resistance Rth
  netlist = add_element(netlist, thermal_res(), 'Rth', {'thermal', 't', 'amb'}, {'Rth', 0.058});
  % thermal capacitance Cth
  netlist = add_element(netlist, thermal_cap(), 'Cth', {'thermal', 't', '0'}, {'Cth', 33.6});
  % environment at ambient temperature
  netlist = add_element(netlist, tempsource(), 'Vamb', {'thermal', 'amb', '0'}, {}, {'dc', 300});

  netlist = add_domain_add_ons(netlist, {'thermal', 'abszeronodename', '0'});
  netlist = add_domain_add_ons(netlist, {'electrical', 'groundnodename', 'gnd'});
end
```

**Fig. 5.24:** Code for constructing a multiphysics netlist for system in Fig. 5.23.

from those that are not. Specifically, we design a master equation engine to handle all the structures not associated with physical domains, *e.g.*, parameters, all the non-I/O unknowns, implicit equations, *etc.*. Then it uses components named NIL interfaces to handle the domain-specific parts of the devices, *i.e.*, the I/Os. More specifically, NIL interfaces set up the relations between device's I/Os and system-level unknowns. Each NIL interface is an add-on component to the master equation engine, designed to handle the NIL in one physical domain. In this way, MAPP can be conveniently extended to support new physical disciplines, simply by adding new types of NILs at the device level, and the corresponding NIL interfaces for the master equation engine, without changing the device's ModSpec format or simulation algorithms.

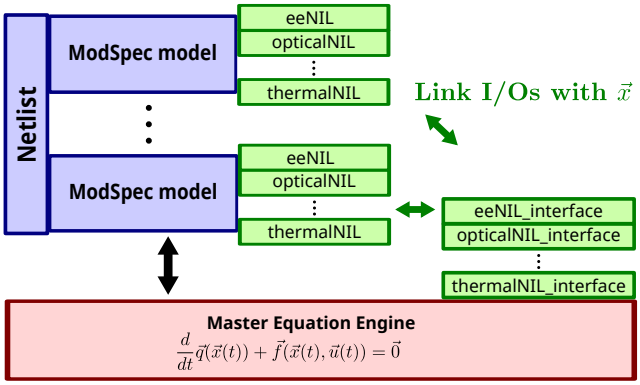



**Fig. 5.25:** MAPP uses a master equation engine with multiple NIL interfaces to construct system-level DAEs from device models and netlist structure.

With the help of the master equation engine and NIL interfaces, MAPP now supports all the "natures" defined in Verilog-A's standard `disciplines.vams` file, including disciplines labelled as electrical, magnetic, thermal, kinematic, *etc.*. All these disciplines are modelled with potentials and flows.

MAPP's multiphysics modelling and simulation capabilities are considerably more general and flexible than those of Verilog-A. Specifically, systems unknowns do not have to be

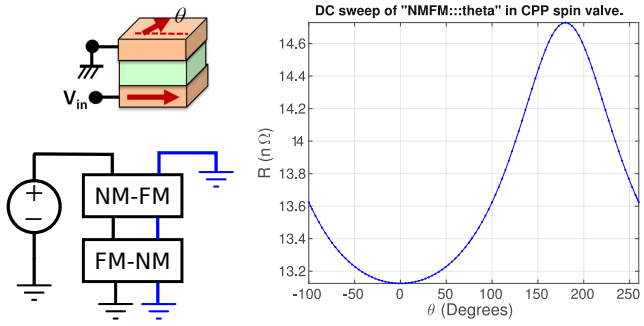



**Fig. 5.26:** DC sweep on a spin valve system in MAPP. The network contains both electrical and spintronic connections (blue wires). Device diagram is adapted from [3].

| **domain NILs** | **I/Os** | **network connectivity** |
|---|---|---|
| optical NIL | incoming/outgoing waves (vector of complex numbers) | two optical ports are connected at each node, the incoming waves on one side are equal to the outgoing waves on the other, and vice-versa. |
| spin NIL | spin voltages and currents (size-3 vectors) | spin "KCLs" and "KVLs" in vector form |
| chemical NIL | concentrations ($C$) and reaction rates ($R$) (scalars) | node connections represent concentrations of reactants; the sum of reaction rates at a node is equal to the change in corresponding concentration, *i.e.*, $\sum R = -\frac{d}{dt}C$. |

**Table 5.1:** NIL descriptions of some selected physical domains supported in MAPP.

potentials or flows. Through the use of Network Interface Layers (NILs) [162] in ModSpec, we allow users to define physical quantities that are suitable and intuitive in their underlying physical domains, especially where circuit laws are not applicable. By writing NIL interfaces for a master equation engine, users can also define system connections, *i.e.*, they specify the multiphysics equivalents of circuit laws by controlling how DAEs are constructed. These interfaces can be easily written by implementing a few API functions that specify how to link NIL quantities to system-level DAE unknowns and equations; they are not limited to using only MNA. Moreover, we leverage MATLAB®'s support for complex numbers and vectors in MAPP, simplifying system connections significantly compared with Verilog-A implementations [3, 160].

Tab. 5.1 summarizes MAPP's formulations for some selected physical domains. Among them, optical connections are modelled with incoming/outgoing waves, which can be thought of as time-varying phasors capturing the modulated envelopes of light. Variables at each optical ports consist of vectors of complex numbers representing these phasors at all simulation frequencies. There is no direct circuit analogy to this formulation. In chemical reaction networks, each node represents the concentration of a reactant. "KCL" at a node involves both the summation of reaction rates and a differential term for the node concentration. Using our infrastructure, such "differential-KCLs" can be defined in MAPP conveniently. To illustrate the simulation capabilities of MAPP on these multiphysics

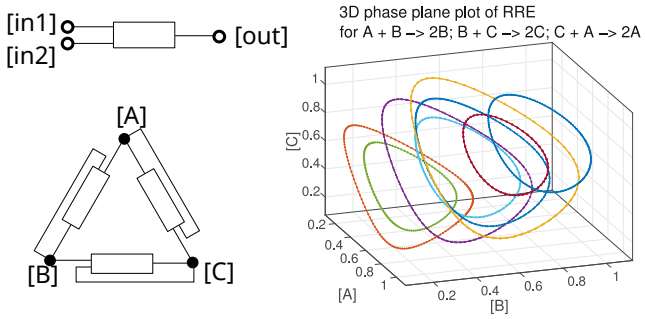



**Fig. 5.27:** Diagrams of a 2-to-1 chemical reaction, a network containing three of such reactions. Transient simulations with different initial concentrations (plotted in 3-D) show that the reaction network constitutes a chemical oscillator.

systems, we run DC and transient analyses on a spin valve and a chemical oscillator in MAPP, and show the results in Fig. 5.26 and Fig. 5.27 respectively.

## 5.4 Prototyping Simulation Algorithms in MAPP

MAPP comes with features that also make prototyping simulation algorithms quick and convenient:

- Simulation algorithms in MAPP rely only on the API of MAPP's DAE objects, as noted earlier; this API does not expose any details of device models or network formulations. This makes it possible, and easy, to write powerful algorithms that apply immediately to any system or device. Device models and Equation Engine code do not need to be modified, or even consulted, when algorithms are written or updated. Various *ad-hoc* concepts used in traditional circuit simulators, such as SPICE's Norton-Theorem-based RHS [154], companion equivalent circuits, *etc.*, are eliminated, making algorithms much more simple and elegant.

- All simulation algorithms in MAPP are object-oriented. The encapsulation induced by such object-oriented implementations enforces modularity, improves code readability and simplifies code documentation. It also enables inheritance between analyses, *i.e.*, developers of higher-level algorithms can easily leverage more basic ones using only a few lines of code. Fig. 5.28 depicts how MAPP's simulation algorithms are structured — starting from the basic numerical routines shown on the top of Fig. 5.28, many algorithms, from the standard DC, AC, transient analyses to more advanced ones, build on others hierarchically. This feature greatly reduces the time and trouble it takes to develop or prototype advanced new algorithms, allowing MAPP to easily support more of them than most other simulators. Over the years, we have prototyped and reproduced many simulation algorithms in our group [163, 194–196].

To illustrate these features of MAPP, we use the shooting algorithm as an example. The

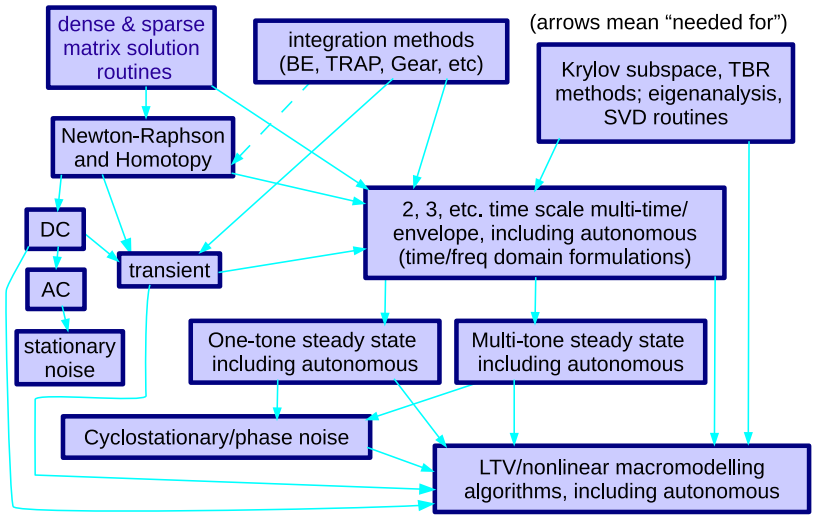



**Fig. 5.28:** Structuring of MAPP's simulation algorithms.

shooting method (henceforth just "shooting") [197] is a numerical algorithm for finding Periodic Steady-State (PSS) responses of systems. Shooting poses the problem of finding a periodic response as that of finding an (initially unknown) initial condition that evolves to itself after one period. In other words, denoting the initial condition by $\vec{x}_0$, shooting can be written as

$$\vec{g}(\vec{x}_0) \triangleq \vec{x}(T) - \vec{x}_0 = \vec{0}, \tag{5.21}$$

where $\vec{x}(t)$ is the solution of the system DAE with initial condition $\vec{x}_0$, *i.e.*, $\vec{x}(t)$ satisfies

$$\frac{d}{dt}\vec{q}(\vec{x}(t)) + \vec{f}(\vec{x}(t), \vec{u}(t)) = \vec{0}, \tag{5.22}$$

$$\text{and} \quad \vec{x}(0) = \vec{x}_0. \tag{5.23}$$

$\vec{g}(\vec{x}_0)$ is an algebraic function of $\vec{x}_0$; as such, it can be solved using numerical algorithms for nonlinear algebraic systems, such as the Newton-Raphson (NR) method [55]. Since evaluating $\vec{g}(\cdot)$ at each NR step involves running a transient simulation, shooting, in essence, reduces the PSS problem (a boundary value problem) to a few initial value problems.

Algorithm 1, showing pseudo-code for MAPP's implementation of shooting, further highlights how algorithm prototyping is quick and convenient in MAPP:

- Shooting in MAPP is formulated using DAEs and is unrelated to device models or physical domains.

- Shooting requires running transient simulations. But since algorithm implementations in MAPP are object-oriented, transient analysis does not need to be re-implemented within shooting. Instead, a transient analysis object is initiated and its methods called. Likewise, shooting itself is also written in an object-oriented manner, with its numerical routines

**Algorithm 1** Shooting Algorithm in MAPP (pseudo-code)

```
shootObj = shoot(DAE): // constructor
 1: shootObj.DAE = DAE;
 2: shootObj.tranObj = LMS(DAE); // transient simulation object
 3: set up member functions: .solve, .g, and .J;
 4: return shootObj;

shootObj.solve (initguess, T):
   x0 ← NR(@g, @J, initguess);
   shootSols = tranObj.solve(x0, 0, T);
   return shootSols;

shootObj.g (x0):
   tranSols = tranObj.solve(x0, 0, T);
   return gout = tranSols(:, n) - x0;

shootObj.J (x0):
   tranSols = tranObj.solve(x0, 0, T);
   Ci_pre = DAE.dq_dx(x0);
   M = eye(n);
   for i = 2:n do
       x = tranSols(:, i); u = inputs(:, i);
       Ci = DAE.dq_dx(x); Gi = DAE.df_dx(x, u);
       M = (Ci + (tpts(i) - tpts(i-1)) * Gi) \ Ci_pre * M;
       Ci_pre = Ci;
   end for
   return Jout = M - eye(n);
```

encapsulated in its member functions. So other analyses, if needed, can also internally use shooting conveniently.

- MAPP's implementation of shooting leverages MATLAB's vector (line 5 in shootObj.J) and sparse matrix (line 7 in shootObj.J) data types and associated functions. This makes the actual code almost as simple as the pseudo-code shown in Algorithm 1.

Shooting was implemented, debugged and tested on a number of circuits in about a week by one of the authors. By way of comparison, it had taken a bright, hard-working graduate student two years to implement shooting in SPICE3 [198]; the implementation was not widely released because the student was unable to debug it satisfactorily before he graduated [199]. In fact, more than twenty years later, shooting is still not widely available in open-source simulators today.

## 5.5 Using MAPP for Oscillator-based Computation

The design and analysis of oscillator-based systems has always been challenging. In particular, standard SPICE transient simulation is often not well suited for capturing oscillator phases accurately. Periodic steady state (PSS) analyses, often only available in commercial simulators, still lack the ability to predict injection locking efficiently. Such analysis is non-trivial for a single oscillator, and becomes much more difficult for coupled oscillator systems.

MAPP has been an indispensable tool in our study of oscillator-based computational

systems. It provides us with efficient simulation capabilities and convenient visualization facilities — both are essential to the realization of our systems. Specifically, it helps the design of oscillator-based computational systems in the following ways.

- **Extraction of oscillator PPVs.** Sec. 5.4 uses the shooting method as an example to show how advanced simulation algorithms can be conveniently implemented in MAPP. Shooting is one method for calculating an oscillator's PSS and is thus immediately useful for oscillator analysis. MAPP has both the shooting method and harmonic balance (including the multi-tone version) available. Building on them, we have also implemented in MAPP both frequency-domain [59] and time-domain [58] methods to calculate an oscillator's PPV numerically. The numerical calculation of PPV is the foundation of phase-base analyses, and is not commonly available in existing simulators.

- **Adlerization of oscillator phase-macromodels.** MAPP can also be used to calculate the GAE model for analyzing injection locking, by numerically integrate over the $t_2$ dimension in (2.15). Based on GAE equations, especially the phase interference function, MAPP can also construct the generalized Kuramoto model for coupled oscillators in the analysis of an OIM system.

- **Prediction of SHIL.** Based on the GAE model, MAPP can numerically predict whether IL and SHIL will happen, and when it does, estimate the locking range and locking phase error.

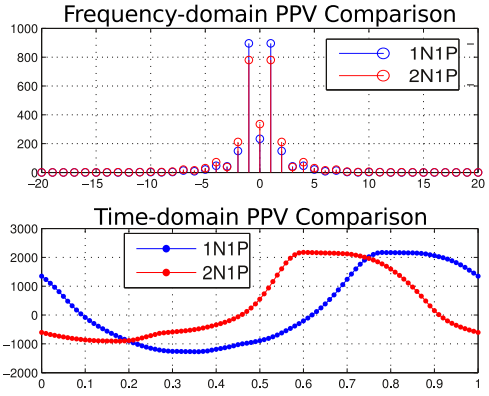



**Fig. 5.29:** PPVs extracted by the design tools from ring oscillator latches with 2N1P and 1N1P inverters.

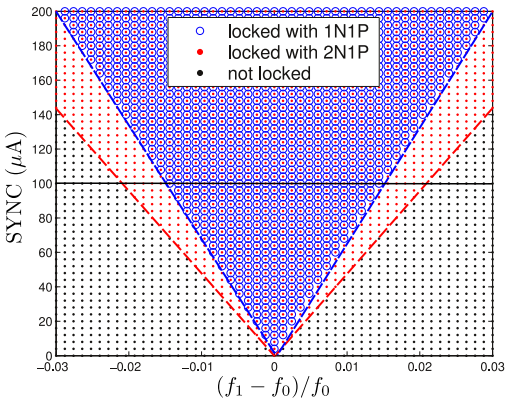



**Fig. 5.30:** Locking range returned by the design tools on ring oscillator latches with 2N1P and 1N1P inverters.

MAPP performs such analysis by calculating the DC operating point of the GAE. When operating points exist, they represent the occurrence of injection locking. And when such DC analysis is performed while we vary the design parameters, the locking range can be obtained. As an example, we apply it to the ring oscillator latch used in Sec. 3.3.1,

and compare the PPV (Fig. 5.29) and SHIL locking range (Fig. 5.30) [147]. From the numerical results, we can easily choose the better design with the larger locking range.

Moreover, the GAE DC solutions gained from MAPP also tell us the locking phase error within the locking range [147] — the difference between the ideal locking phase $\Delta\hat{\phi}_i$ assuming no variability in the oscillator's frequency and the oscillator's actual locked phase $\Delta\phi_i$. As an example, Fig. 5.31 shows the results obtained on the three-stage ring oscillator used in Sec. 3.3.1.

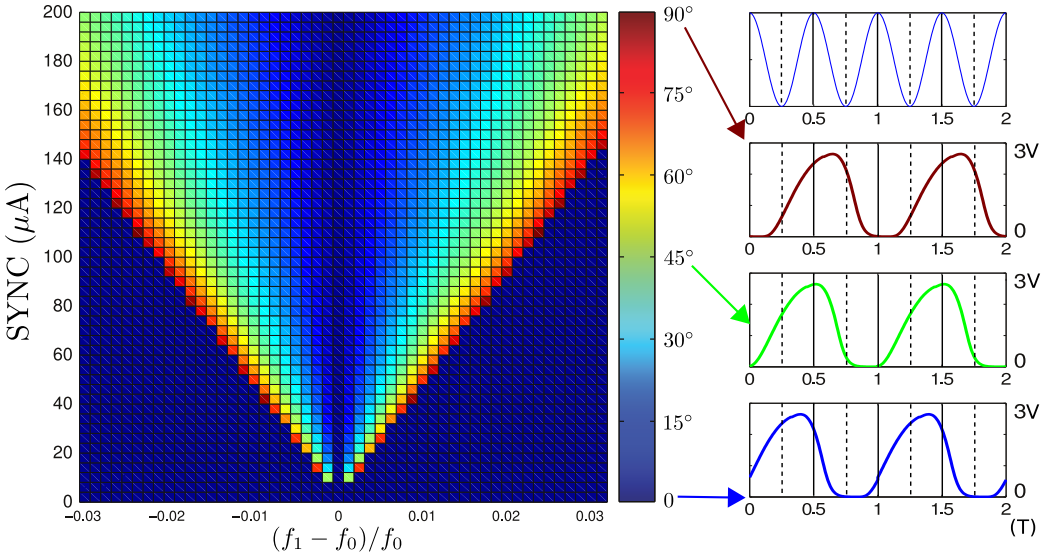



**Fig. 5.31:** Locking phase errors $|\Delta\hat{\phi}_i - \Delta\phi_i|$ within the locking range.

### ○ Full System Transient Simulation with Phase Macromodels

In the operation of a phase logic FSM, when the logic values of inputs change over time, *i.e.*, their phase differences with respect to the reference signal shift back and forth between 0 and 180°, designers need to know whether the system is indeed performing computation properly with phase-based logic.

For this purpose, traditional transient simulation can be performed on the DAE of the full system:

$$\frac{d}{dt}\vec{q}_{\text{full}}(\vec{x}_{\text{full}}) + \vec{f}_{\text{full}}(\vec{x}_{\text{full}}) + \vec{b}_{\text{full}}(t) = \vec{0}. \tag{5.24}$$

However, if we separate system unknowns $\vec{x}_{\text{full}}$ into $\vec{x}_{\text{osc}_i}$ and $\vec{x}_{\text{other}}$, where $\vec{x}_{\text{osc}_i}, i = 1, 2, \cdots, k$ represents unknowns inside each of the $k$ oscillator latches, we know $\vec{x}_{\text{osc}_i}$ can be approximated well with its steady state response $\vec{x}_{\text{osc}_i(\text{s})}$ and its phase $\alpha_i$ as in ((2.2)). As $\alpha_i$ is unbounded in simulation, similar to the formulation of GAE, we use the locking phase error $\Delta\phi_i = f_0 t + f_0\alpha_i(t) - f_1 t$ instead, such that

$$\vec{x}_{\text{osc}_i}(t) = \vec{x}_{\text{osc}_i(\text{s})}((f_1 t + \Delta\phi_i(t))/f_0), \tag{5.25}$$

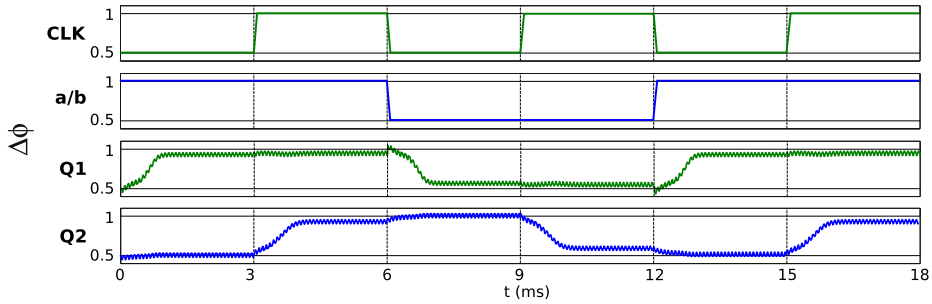



**Fig. 5.32:** Transient simulation of the serial adder in Fig. 3.11 with oscillator latches replaced by their PPV macromodels. $\Delta\phi = 0.5$ indicates opposite phase w.r.t. REF, thus encoding phase-based 0, while $\Delta\phi = 1$ encodes 1.

where $\Delta\phi_i$ is governed by

$$\frac{d}{dt}\Delta\phi_i(t) = f_0 - f_1 + f_0 \cdot \vec{v}_i^T\left(T_1\Delta\phi_i(t) + t\right) \cdot \vec{b}_i(t), \tag{5.26}$$

where external $\vec{b}_i(t)$ can be calculated from $\vec{x}_{\text{other}}$ based on circuit connections.

The formulation in (5.26) is slightly different from the GAE in ((2.20)) in that the fast varying mode is preserved instead of averaged out [81]. In this way, the full system can be formulated as

$$\frac{d}{dt}\vec{q}'(\vec{x}_{\text{other}}, \Delta\vec{\phi}) + \vec{f}'(\vec{x}_{\text{other}}, \Delta\vec{\phi}) + \vec{b}(t) = \vec{0}, \tag{5.27}$$

where $\Delta\vec{\phi}$ represents all $\Delta\phi_i$s. For each oscillator latch, all its voltage and current unknowns are now represented by only one scalar. Therefore, the system size is reduced.

Instead of formulating (5.27) analytically, our design tools allow users to identify subcircuits that describe oscillator latches and replace them with their PPV macromodels as in (5.26). Since the PPV of the latch has already been calculated in previous stages of design, little computation is introduced in reformulating the full system into (5.27). Such reformulation with PPV macromodels results in an equation system not only with less unknowns, but also with unknowns that are more directly related to the phase operation of the system.

As an example of this full-system analysis, we simulate the operation of the serial adder in Fig. 3.11 while replacing the two oscillator latches with their phase-domain models. From the transient simulation results in Fig. 5.32, we can observe the phase transitions of the two latches when the adder is working, showing how the slave latch follows the master's phase in the flip-flop.

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

In this dissertation, we introduced novel schemes of computing using coupled oscillator systems, as well as the analyses and tools that have made them possible.

We showed that oscillators can be used to implement phase-encoded finite state machines for general-purpose Boolean computation, extending a scheme originally proposed by Goto and von Neumann in the 1950s. We also demonstrated oscillator-based Ising machines, suitable for solving combinatorial optimization problems efficiently. We presented the mechanisms of our schemes, explored their practical design, and demonstrated working hardware prototypes as proof of concept. These new paradigms greatly expanded the scope of oscillator-based computing.

The ideas we presented are original and innovative. To the best of our knowledge, our work demonstrated the world's *first oscillator-based finite state machine*, and the *first practical Ising machine* based on self-sustaining oscillators. Moreover, our exploration of non-electronic oscillators resulted in the *first demonstration of sub-harmonic injection locking in mechanical metronomes*, which has led to the *first oscillator-based mechanical memory* with phase-based encoding.

## 6.2 Future Work

We have also identified several directions for future work/research.

- **Integrated circuits (ICs) for PHLOGON and OIM.** One immediate direction is to scale up our proof-of-concept implementations. As our existing prototypes use standard CMOS circuit technology, moving their implementations from breadboards/PCBs onto ICs does not represent a significant technological challenge. On ICs, Ising machines with tens of thousands or even a hundred thousand oscillator spins can significantly surpass any existing Ising machine in scale. Such Ising chips can be directly plugged into computers (through interfaces such as USB and PCI); CPUs can offload previously intractable Ising problems onto them and get high-quality solutions with ultra-low power consumption and high speed. These chips can be useful both for data centers and for edge computing on smartphones and IoT devices. This future direction is a low-hanging fruit with immediate wide-ranging impacts.

- **Nano-oscillator-based implementations of PHLOGON and OIM.** As mentioned in this dissertation, the mechanism for the proposed systems is general for all oscillators, and the implementations are not limited to using CMOS technology. Various nano-oscillators, implemented with optics, MEMS, spintronics, memristive devices, *etc.*, have potential for further improving the speed, compactness and energy efficiency of our computational systems. We expect to soon see efforts on the demonstration of Ising machines using spin-torque nano-oscillators, memristive-device-based relaxation oscillators, *etc.*, from the device physics community. While such small-scale hardware demonstrations may be around the corner, their practical large-scale implementation and deployment remain a challenge, due to the progress of the underlying device technology. In any scenario, the device compact modelling flow, the multiphysics capabilities, and the oscillator-specific analyses in our design tool MAPP can greatly facilitate this exploration in the future.

- **Detailed design trade-offs for PHLOGON.** Our preliminary work studies these trade-offs using circuit theory and simulation analysis; hardware-based validations and explorations are needed as future work.

- **Scaling analysis for OIM.** Our Ising machines do not solve NP-hard problems in polynomial time — no Ising machines do. Preliminary results show that even though the time to solution does not increase much with the problem size, in order to achieve the global optimum, the number of samples needed from the machines grows exponentially. How the solution quality degrades with problem size, and how by allowing more time in OIM's annealing schedule does the quality improve, are both questions that require more in-depth analysis. Also, it is worth noting that in most real-world scenarios, achieving the global optimum of optimization problems is not necessary; it is usually good enough to get high-quality local optima very fast. As a physical system directly performing the optimization, OIM is naturally suitable for this purpose.

- **Maintaining the open-source releases of PHLOGON, OIM and MAPP.** We have always strived to make our work publicly available under open-source licenses for reproducible research. Over the years, we have been releasing our code and hardware designs by hosting snapshots on our websites, setting up our own GitLab servers, and most recently using GitHub repositories. Maintaining the public releases for PHLOGON, OIM and MAPP is also an important part of future work.

## Bibliography

[1] A. Marandi, Z. Wang, K. Takata, R. L. Byer and Y. Yamamoto. Network of time-multiplexed optical parametric oscillators as a coherent Ising machine. *Nature Photonics*, 8(12):937–942, 2014.

[2] C. Helmberg, F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10(3):673–696, 2000.

[3] K. Camsari, S. Ganguly, and S. Datta. Modular approach to spintronics. *Scientific reports*, 5.

[4] G. E. Moore et al. Cramming More Components onto Integrated Circuits, 1965.

[5] L. Wilson. International Technology Roadmap for Semiconductors (ITRS). 2013.

[6] C. Hu. *Modern semiconductor devices for integrated circuits*, volume 2. Prentice Hall Upper Saddle River, NJ, 2010.

[7] H.L. Hennessy, D.A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[8] D. Hisamoto, W. C. Lee, J. Kedzierski, H. Takeuchi, K. Asano, C. Kuo, E. Anderson, T.-J. King, J. Bokor, and C. Hu. FinFET—a Self-aligned Double-gate MOSFET Scalable to 20 nm. *IEEE Transactions on Electron Devices*, 47(12):2320–2325, 2000.

[9] N. Singh, A. Agarwal, L.K. Bera, T.Y Liow, R. Yang, S.C. Rustagi, C.H. Tung, R. Kumar, G.Q. Lo, N. Balasubramanian, and others. High-performance fully depleted silicon nanowire (diameter/spl les/5 nm) gate-all-around CMOS devices. *IEEE Electron Device Letters*, 27(5):383–386, 2006.

[10] N.P. Jouppi, C. and Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A.Borchers, and others. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12. IEEE, 2017.

[11] J.M. Shalf, R. Leland. Computing beyond moore's law. *Computer*, 48(SAND-2015-8039J), 2015.

[12] C. Sun, M.T. Wade, Y. Lee, and J.S. Orcutt, L. Alloatti, M.S. Georgas, A.S. Waterman,J.M. Shainline, R.R. Avizienis, S. Lin, and others. Single-chip microprocessor that communicates directly using light. *Nature*, 528(7583):534, 2015.

[13] S. Tehrani, J.M. Slaughter, E. Chen, M. Durlam, J. Shi, M. DeHerren. Progress and outlook for MRAM technology. *IEEE Transactions on Magnetics*, 35(5):2814–2819, 1999.

[14] K. S. Novoselov, V. I. Fal, L. Colombo, P. R. Gellert, M. G. Schwab, K. Kim, et al. A roadmap for graphene. *Nature*, 490(7419):192–200, 2012.

[15] M. M. Shulaker, G. Hills, N. Patil, H. Wei, H. Chen, H.-S. P. Wong, and S. Mitra. Carbon nanotube computer. *Nature*, 501(7468):526–530, 2013.

[16] J. Deng, K. Ryu, C. Zhou, et al. Carbon nanotube transistor circuits: Circuit-level performance benchmarking and design options for living with imperfections. In *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, pages 70–588, 2007.

[17] A. Singh, H. A. Zeineddine, A. Aziz, S. Vishwanath, et al. A heterogeneous CMOS-CNT architecture utilizing novel coding of boolean functions. In *Proceedings of the 2007 IEEE International Symposium on Nanoscale Architectures*, pages 15–20. IEEE Computer Society, 2007.

[18] H. Arsenault. *Optical processing and computing*. Elsevier, 2012.

[19] B. Behin-Aein, D. Datta, S. Salahuddin, and S. Datta. Proposal for an all-spin logic device with built-in memory. *Nature nanotechnology*, 5(4):266–270, 2010.

[20] A.K. Yadav, K.X. Nguyen, Z. Hong, P. García-Fernández, P. Aguado-Puente, C.T. Nelson, S. Das, B. Prasad, D. Kwon, Daewoong, and others. Spatially resolved steady-state negative capacitance. *Nature*, 565(7740):468, 2019.

[21] P. Sheridan, K.-H. Kim, S. Gaba, T. Chang, L. Chen and W. Lu. Device and SPICE modeling of RRAM devices. *Nanoscale*, 3(9):3833–3840, 2011.

[22] S. Lai. Current status of the phase change memory and its future. In *IEEE International Electron Devices Meeting 2003*, pages 10–1. IEEE, 2003.

[23] H.-S. P. Wong, S. and Raoux, S. Kim, J. Liang, J.P. Reifenberg, B. and Rajendran, M. Asheghi, K.E. Goodson. Phase change memory. *Proceedings of the IEEE*, 98(12):2201–2227, 2010.

[24] M. Kund, G. Beitel, C.-U. Pinnow, T. Rohr, J. Schumann, R. Symanczyk, K. Ufert, and G. Muller. Conductive bridging RAM (CBRAM): An emerging non-volatile memory technology scalable to sub 20nm. In *IEEE International Electron Devices Meeting, 2005. IEDM Technical Digest.*, pages 754–757. IEEE, 2005.

[25] C. Gopalan, Y. Ma, T. Gallo, J. Wang, E. Runnion, J. Saenz, F. Koushan, P. Blanchard, S. Hollmer, S. Demonstration of conductive bridging random access memory (CBRAM) in logic CMOS process. *Solid-State Electronics*, 58(1):54–61, 2011.

[26] F. Arute, K. Arya, R. Babbush, D. Bacon, J.C. Bardin, R. Barends, R. Biswas, S. Boixo, and others. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[27] Z. Wang, S. Joshi, S.E. Savel'ev, H. Jiang, R. Midya, P. Lin, M. Hu, N. Ge, J.P. Strachan, Z. Li, and others. Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing. *Nature materials*, 16(1):101, 2017.

[28] T. Wang and J. Roychowdhury. PHLOGON: PHase-based LOGic using Oscillatory Nanosystems. In *Proc. UCNC*, LNCS sublibrary: Theoretical computer science and general issues. Springer, July 2014. DOI link.

[29] T. Wang and J. Roychowdhury. Oscillator-based Ising Machine. *arXiv preprint arXiv:1709.08102*, 2017.

[30] **(Best Paper)** T. Wang and J. Roychowdhury. OIM: Oscillator-based Ising Machines for Solving Combinatorial Optimisation Problems. In *Proc. UCNC*, LNCS sublibrary: Theoretical computer science and general issues. Springer, June 2019. Preprint available at arXiv:1903.07163 [cs.ET].

[31] T. Wang, L. Wu, J. Roychowdhury. New Computational Results and Hardware Prototypes for Oscillator-based Ising Machines. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*, pages 239:1–239:2, 2019.

[32] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558, 1982.

[33] L. O. Chua and L. Yang. Cellular neural networks: Applications. *Circuits and Systems, IEEE Transactions on*, 35(10):1273–1290, 1988.

[34] T. Shibata, R. Zhang, S. P. Levitan, D. E. Nikonov and G. I. Bourianoff. CMOS supporting circuitries for nano-oscillator-based associative memories. In *International Workshop on Cellular Nanoscale Networks and Their Applications (CNNA)*, pages 1–5. IEEE, 2012.

[35] S. P. Levitan, Y. Fang, D. H. Dash, T. Shibata, D. E. Nikonov and G. I. Bourianoff. Non-Boolean associative architectures based on nano-oscillators. In *International Workshop on Cellular Nanoscale Networks and Their Applications (CNNA)*, pages 1–6. IEEE, 2012.

[36] P. Maffezzoni, B. Bahr, Z. Zhang and L. Daniel. Analysis and design of weakly coupled LC oscillator arrays based on phase-domain macromodels. *IEEE Trans. CAD*, 34(1):77–85, 2015.

[37] P. Maffezzoni, B. Bahr, Z. Zhang and L. Daniel. Oscillator array models for associative memory and pattern recognition. *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, 62(6):1591–1598, June 2015.

[38] F. Hoppensteadt and E. Izhikevich. Synchronization of laser oscillators, associative memory, and optical neurocomputing. *Physical Review E*, 62(3):4010, 2000.

[39] F. Hoppensteadt and E. Izhikevich. Pattern recognition via synchronization in phase-locked loop neural networks. *IEEE Transactions on Neural Networks*, 11(3):734–738, 2000.

[40] T. Nishikawa, F. Hoppensteadt and Y.-C. Lai. Oscillatory associative memory network with perfect retrieval. *Physica D: Nonlinear Phenomena*, 197(1):134–148, 2004.

[41] G. Csaba, M. Pufall, D. E. Nikonov, G. I. Bourianoff, A. Horvath, T. Roska and W. Porod. Spin torque oscillator models for applications in associative memories. In *International Workshop on Cellular Nanoscale Networks and Their Applications (CNNA)*, pages 1–2. IEEE, 2012.

[42] G. Csaba and W. Porod. Computational study of spin-torque oscillator interactions for non-Boolean computing applications. *IEEE Transactions on Magnetics*, 49(7):4447–4451, 2013.

[43] M. Sharad, D. Fan and K. Roy. Energy-Efficient and Robust Associative Computing With Injection-Locked Dual-Pillar Spin-Torque Oscillators. *IEEE Transactions on Magnetics*, 51(7):1–9, 2015.

[44] G. Csaba, A. Papp, W. Porod and R. Yeniceri. Non-boolean computing based on linear waves and oscillators. In *European Solid State Device Research Conference (ESSDERC)*, pages 101–104. IEEE, 2015.

[45] T. C. Jackson, A. A. Sharma, J. A. Bain, J. A. Weldon and L. Pileggi. Oscillatory Neural Networks Based on TMO Nano-Oscillators and Multi-Level RRAM Cells. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 5(2):230–241, 2015.

[46] A. A. Sharma, T. C. Jackson, M. Schulaker, C. Kuo, C. Augustine, J. A. Bain, H.-S. P. Wong, S. Mitra, L. Pileggi and J. A. Weldon. High performance, integrated 1T1R oxide-based oscillator: Stack engineering for low-power operation in neural network applications. In *Symposium on VLSI Technology (VLSI Technology)*, pages T186–T187. IEEE, 2015.

[47] S. Datta, N. Shukla, M. Cotter, A. Parihar and A. Raychowdhury. Neuro inspired computing with coupled relaxation oscillators. In *Proc. IEEE DAC*, pages 1–6. ACM, 2014.

[48] P. Maffezzoni, L. Daniel, N. Shukla, S. Datta, A. Raychowdhury and V. Narayanan. Modelling hysteresis in vanadium dioxide oscillators. *IET Electronics Letters*, 51(11):819–820, 2015.

[49] P. Maffezzoni, L. Daniel, N. Shukla, S. Datta and A. Raychowdhury. Modeling and Simulation of Vanadium Dioxide Relaxation Oscillators. *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, 62(9):2207–2215, 2015.

[50] X. Lai and J. Roychowdhury. Fast Simulation of Large Networks of Nanotechnological and Biochemical Oscillators for Investigating Self-Organization Phenomena. In *Proc. IEEE ASP-DAC*, pages 273–278, January 2006. DOI link.

[51] T. Yang, R.A. Kiehl, L.O. Chua. Tunneling phase logic cellular nonlinear networks. *International Journal of Bifurcation and chaos*, 11(12):2895–2911, 2001.

[52] K. Yogendra, C. Liyanagedera, D. Fan, Y. Shim and K. Roy. Coupled Spin-Torque Nano-Oscillator-Based Computation: A Simulation Study. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(4):56, 2017.

[53] D. Wang. A comparison of CNN and LEGION networks. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 3, pages 1735–1740. IEEE, 2004.

[54] K. Chen and D. Wang. A dynamically coupled neural oscillator network for image segmentation. *Neural Networks*, 15(3):423–439, 2002.

[55] J. Roychowdhury. Numerical simulation and modelling of electronic and biochemical systems. *Foundations and Trends in Electronic Design Automation*, 3(2-3):97–303, December 2009.

[56] A. Demir, A. Mehrotra, and J. Roychowdhury. Phase Noise in Oscillators: a Unifying Theory and Numerical Methods for Characterization. *IEEE Trans. Ckts. Syst. – I: Fund. Th. Appl.*, 47:655–674, May 2000. DOI link.

[57] J. Roychowdhury. Exact analytical equations for predicting nonlinear phase errors and jitter in ring oscillators. In *Proc. IEEE International Conference on VLSI Design*, January 2005.

[58] A. Demir and J. Roychowdhury. A Reliable and Efficient Procedure for Oscillator PPV Computation, with Phase Noise Macromodelling Applications. *IEEE Trans. CAD*, pages 188–197, February 2003. DOI link.

[59] T. Mei and J. Roychowdhury. PPV-HB: Harmonic Balance for Oscillator/PLL Phase Macromodels. In *Proc. ICCAD*, pages 283–288, Nov. 2006.

[60] J. Roychowdhury. *Applied and Computational Control, Signals and Circuits – Recent Developments*, chapter 3 (Multi-time PDEs for Dynamical System Analysis), pages 85–143. Kluwer Academic, 2001.

[61] R. Adler. A study of locking phenomena in oscillators. *Proc. IEEE*, 61:1380–1385, 1973. Reprinted from [200].

[62] J. A. Acebrón, L. L. Bonilla, C. J. P. Vicente, F. Ritort and R. Spigler. The Kuramoto Model: A Simple Paradigm for Synchronization Phenomena. *Reviews of Modern Physics*, 77(1):137, 2005.

[63] Y. Kuramoto. Self-entrainment of a population of coupled non-linear oscillators. In *International symposium on mathematical problems in theoretical physics*, pages 420–422. Springer, 1975.

[64] Steven Strogatz. *Sync: The Emerging Science of Spontaneous Order*. Theia, March 2003.

[65] D. Cumin, C.P. Unsworth. Generalising the Kuramoto model for the study of neuronal synchronisation in the brain. *Physica D: Nonlinear Phenomena*, 226(2):181–196, 2007.

[66] D. Hansel, G. Mato, C. Meunier. Phase dynamics for weakly coupled Hodgkin-Huxley neurons. *EPL (Europhysics Letters)*, 23(5):367, 1993.

[67] H. Sakaguchi, Y. Kuramoto. A soluble active rotater model showing phase transitions via mutual entertainment. *Progress of Theoretical Physics*, 76(3):576–581, 1986.

[68] A. M. Lyapunov. The General Problem of the Stability of Motion. *International Journal of Control*, 55(3):531–534, 1992.

[69] R. L. Wigington. A New Concept in Computing. *Proceedings of the Institute of Radio Engineers*, 47:516–523, April 1959. DOI link.

[70] John von Neumann. Non-linear capacitance or inductance switching, amplifying and memory devices, 1954. Basic paper for Patent 2,815,488, filed April 28, granted December 3, 1957, and assigned to IBM. Reprinted in [71, Paper 11, pp. 379–419]. Google patents link.

[71] A. H. Taub, editor. *John von Neumann: Collected Works. Volume V: Design of Computers, Theory of Automata and Numerical Analysis*. Pergamon Press, New York, NY, USA, 1963. See also volumes I–IV, VI [201–205].

[72] J. M. Manley and R. E Rowe. Some general properties of nonlinear elements – Part I. General energy relations. *Proceedings of the Institute of Radio Engineers*, 44(7):904–913, July 1956.

[73] Eiichi Goto. New Parametron circuit element using nonlinear reactance. *KDD Kenyku Shiryo*, October 1954.

[74] S. Oshima. Introduction to Parametron. *Denshi Kogyo*, 4(11):4, December 1955.

[75] Oshima, Enemoto, and Watanabe. Oscillation theory of Parametron and method of measuring nonlinear elements. *KDD Kenkyu Shiryo*, November 1955.

[76] H. Takahashi. The Parametron. *Tsugakkat Shi*, 39(6):56, June 1956.

[77] S. Muroga. Elementary principle of Parametron and its application to digital computers. *Datamation*, 4(5):31–34, September/October 1958.

[78] IPSJ Computer Museum: Parametron (web page). http://museum.ipsj.or.jp/en/computer/dawn/0007.html.

[79] IPSJ Computer Museum: PC-1 Parametron Computer (web page). http://museum.ipsj.or.jp/en/computer/dawn/0016.html.

[80] Willy Hoe and Eiichi Goto. *Quantum Flux Parametron: A Single Quantum Flux Superconducting Logic Device*, volume 2 of *Studies in Josephson Supercomputers*. World Scientific, 1991.

[81] P. Bhansali and J. Roychowdhury. Gen-Adler: The generalized Adler's equation for injection locking analysis in oscillators. In *Proc. IEEE ASP-DAC*, pages 522–227, January 2009. DOI link.

[82] A. Neogy and J. Roychowdhury. Analysis and Design of Sub-harmonically Injection Locked Oscillators. In *Proc. IEEE DATE*, Mar 2012. DOI link.

[83] Andrei Slavin. Microwave sources: Spin-torque oscillators get in phase. *Nature Nanotechnology*, 4(8):479–480, Aug 2009.

[84] Peter de Groot and Hans Fangohr. Spin Torque and Dynamics in Magnetic Nanostructures (web page). http://www.icss.soton.ac.uk/research/nano.html.

[85] R. A. Kiehl and T. Ohshima. Bistable locking of single-electron tunneling elements for digital circuitry. *Appl. Phys. Lett.*, 67(17):2494–2496, Oct 1995.

[86] T. Li and R. A. Kiehl. Operating regimes for multivalued single-electron tunneling phase logic. *Journal of Applied Physics*, 93:9291–9297, June 2003.

[87] T. Ohshima and R. A. Kiehl. Operation of bistable phase-locked single-electron tunneling logic elements. *J. Appl. Phys.*, April 1996.

[88] R.A. Kiehl and T. Ohshima. Bistable locking of single-electron tunneling elements for digital circuitry. *Applied Physics Letters*, 67:2494–2496, 1995.

[89] Clark T-C. Nguyen. Vibrating RF MEMS for Next Generation Wireless Applications. In *Proc. IEEE CICC*, May 2004.

[90] M. B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, Jan 2000.

[91] T. Wang. Sub-harmonic Injection Locking in Metronomes. *arXiv preprint arXiv:1709.03886*, 2017.

[92] T. Wang. Achieving Phase-based Logic Bit Storage in Mechanical Metronomes. *arXiv preprint arXiv:1710.01056*, 2017.

[93] J. Pantaleone. Synchronization of Metronomes. *American Journal of Physics*, 70(10):992–1000, 2002.

[94] J. Jia, Z. Song, W. Liu, J. Kurths and J. Xiao. Experimental Study of the Triplet Synchronization of Coupled Nonidentical Mechanical Metronomes. *Scientific reports*, 5, 2015.

[95] T. Wang and J. Roychowdhury. Well-Posed Models of Memristive Devices. *arXiv preprint arXiv:1605.04897*, 2016.

[96] T. Wang. Modelling Multistability and Hysteresis in ESD Clamps, Memristors and Other Devices. In *Proc. IEEE CICC*, pages 1–10. IEEE, 2017.

[97] C. Makkar, W.E. Dixon, W.G. Sawyer, and G. Hu. A new continuously differentiable friction model for control systems design. In *Advanced Intelligent Mechatronics. Proceedings, 2005 IEEE/ASME International Conference on*, pages 600–605. IEEE, 2005.

[98] Douglas C. Giancoli. *Physics for Scientists and Engineers*. Prentice-Hall, Englewood Cliff, N.J., 1989.

[99] Mircea Grigoriu. *Stochastic Calculus: Applications in Science and Engineering*. Birkhäuser, Boston, 2002.

[100] David Middleton. *An Introduction to Statistical Communication Theory*. Wiley-IEEE, New York, 1996.

[101] S.O. Toh, Y. Tsukamoto, Z. Guo, L. Jones, T.J.K. Liu, and B. Nikolic. Impact of random telegraph signals on Vmin in 45nm SRAM. In *Proceedings of the IEEE International Electron Devices Meeting*, pages 767–770, 2009.

[102] Y. Tsukamoto, S.O. Toh, C. Shin, A. Mairena, T.J.K. Liu, and B. Nikolic. Analysis of the relationship between random telegraph signal and negative bias temperature instability. In *Proceedings of the IEEE International Reliability Physics Symposium*, pages 1117–1121, 2010.

[103] M. Jamal Deen, Mehdi H. Kazemeini, and Sasan Naseh. Performance characteristics of an ultra-low power VCO. In *Proc. IEEE ISCAS*, May 2003.

[104] S. Farzeen, Guoyan Ren, and Chunhong Chen. An ultra-low power ring oscillator for passive UHF RFID transponders. In *Proc. IEEE MWSCAS*, pages 558–561, Aug 2010.

[105] F. Cilek, K. Seemann, D. Brenk, J. Essel, J. Heidrich, R. Weigel, and G. Holweg. Ultra low power oscillator for UHF RFID transponder. In *Proc. IEEE Freq. Contr. Symp.*, pages 418–421, May 2008.

[106] M. D. Feuer, R. H. Hendel, R. A. Kiehl, J. C. M Hwang, V. G. Keramidas, C. L. Allyn, and R. Dingle. High-speed low-voltage ring oscillators based on selectively doped heterojunction transistors. *IEEE Electron Device Letters*, EDL-4(9):306–307, September 1983.

[107] D. Weinstein and S. A. Bhave. The Resonant Body Transistor. *Nano letters*, 10(4):1234–1237, 2010.

[108] M. R. Pufall, W. H. Rippard, S. Kaka, T. J. Silva, and S. E. Russek. Frequency modulation of spin-transfer oscillators. *Applied Physics Letters*, 86(8):082506–+, February 2005.

[109] S. Kaka, M. R. Pufall, W. H. Rippard, T. J. Silva, S. E. Russek, and J. A. Katine. Mutual phase-locking of microwave spin torque nano-oscillators. *Nature*, 437:389–392, September 2005.

[110] W. H. Rippard, M. R. Pufall, S. Kaka, T. J. Silva, S. E. Russek, and J. A. Katine. Injection locking and phase control of spin transfer nano-oscillators. *Phys. Rev. Lett.*, 95(6):067203, Aug 2005.

[111] T. Wang and J. Roychowdhury. Rigorous Q Factor Formulation and Characterization for Nonlinear Oscillators, 2017.

[112] T. Wang. Analyzing Oscillators using Describing Functions. *arXiv preprint arXiv:1710.02000*, 2017.

[113] E. Ising. Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik A Hadrons and Nuclei*, 31(1):253–258, 1925.

[114] S.G. Brush. History of the Lenz-Ising Model. *Rev. Mod. Phys.*, 39:883–893, Oct 1967.

[115] F. Barahona. On the computational complexity of Ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241, 1982.

[116] P. L. McMahon, A. Marandi, Y. Haribara, R. Hamerly, C. Langrock, S. Tamate and T. Inagaki, H. Takesue, S. Utsunomiya, K. Aihara and others. A fully-programmable 100-spin coherent Ising machine with all-to-all connections. *Science*, page 5178, 2016.

[117] T. Inagaki, Y. Haribara, K. Igarashi, T. Sonobe, S. Tamate, T. Honjo, A. Marandi, P. L. McMahon, T. Umeki, K. Enbutsu and others. A Coherent Ising machine for 2000-node Optimization Problems. *Science*, 354(6312):603–606, 2016.

[118] M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk and others. Quantum Annealing with Manufactured Spins. *Nature*, 473(7346):194, 2011.

[119] Z. Bian, F. Chudak, R. Israel, B. Lackey, W. G. Macready and A. Roy. Discrete optimization using quantum annealing on sparse Ising models. *Frontiers in Physics*, 2:56, 2014.

[120] M. Yamaoka, C. Yoshimura, M. Hayashi, T. Okuyama, H. Aoki and H. Mizuno. A 20k-spin Ising Chip to Solve Combinatorial Optimization Problems with CMOS Annealing. *IEEE Journal of Solid-State Circuits*, 51(1):303–309, 2016.

[121] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[122] A. Lucas. Ising formulations of many NP problems. *arXiv preprint arXiv:1302.5843*, 2013.

[123] I. Mahboob, H. Okamoto and H. Yamaguchi. An electromechanical Ising Hamiltonian. *Science Advances*, 2(6):e1600236, 2016.

[124] K. Y. Camsari, R. Faria, B. M. Sutton and S. Datta. Stochastic p-Bits for Invertible Logic. *Physical Review X*, 7(3):031014, 2017.

[125] Maliheh Aramon, Gili Rosenberg, Elisabetta Valiante, Toshiyuki Miyazawa, Hirotaka Tamura, and Helmut G. Katzgraber. Physics-Inspired Optimization for Quadratic Unconstrained Problems Using a Digital Annealer. In *arXiv:1806.08815 [physics.comp-ph]*, August 2018.

[126] T. Wang and J. Roychowdhury. OIM: Oscillator-based Ising Machines for Solving Combinatorial Optimisation Problems. *arXiv preprint arXiv:1903.07163*, 2019.

[127] T. Wang, L. Wu, J. Roychowdhury. Late Breaking Results: New Computational Results and Hardware Prototypes for Oscillator-based Ising Machines. *arXiv preprint arXiv:1904.10211*, 2019.

[128] Z. Bian, F. Chudak, W.G. Macready, G. Rose. The Ising model: teaching an old problem new tricks. *D-Wave Systems*, 2, 2010.

[129] R. Harris, J. Johansson, A. J. Berkley, M. W. Johnson, T. Lanting, S. Han, P. Bunyk, E. Ladizinsky, T. Oh, I. Perminov and others. Experimental Demonstration of a Robust and Scalable Flux Qubit. *Physical Review B*, 81(13):134510, 2010.

[130] T. F. Rønnow, Z. Wang, J. Job, S. Boixo, S. V. Isakov, D. Wecker, J. M. Martinis, D. A. Lidar and M. Troyer. Defining and detecting quantum speedup. *Science*, 345(6195):420–424, 2014.

[131] V. S. Denchev, S. Boixo, S. V. Isakov, N. Ding, R. Babbush, V. Smelyanskiy, J. Martinis, H. Neven. What is the computational value of finite-range tunneling? *Physical Review X*, 6(3):031015, 2016.

[132] Kenta Takata, Alireza Marandi, Ryan Hamerly, Yoshitaka Haribara, Daiki Maruo, Shuhei Tamate, Hiromasa Sakaguchi, Shoko Utsunomiya, and Yoshihisa Yamamoto. A 16-bit coherent Ising machine for one-dimensional ring and cubic graph problems. *Scientific Reports*, 6:34089, 2016.

[133] B. Sutton, K. Y. Camsari, B. Behin-Aein and S. Datta. Intrinsic optimization using stochastic nanomagnets. *Scientific Reports*, 7, 2017.

[134] K. Yamamoto, W. Huang, S. Takamaeda-Yamazaki, M. Ikebe, T. Asai, M. Motomura. A Time-Division Multiplexing Ising Machine on FPGAs. In *Proceedings of the 8th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, page 3. ACM, 2017.

[135] S. Shinomoto, Y. Kuramoto. Phase transitions in active rotator systems. *Progress of Theoretical Physics*, 75(5):1105–1110, 1986.

[136] L. D. Landau and E. M. Lifshitz. *Statistical Physics: V. 5: Course of Theoretical Physics*. Pergamon press, 1969.

[137] R. Yuan, Y. Ma, B. Yuan and P. Ao. Constructive Proof of Global Lyapunov Function as Potential Function. *arXiv preprint arXiv:1012.2721*, 2010.

[138] T. Myklebust. Solving maximum cut problems by simulated annealing. *arXiv preprint arXiv:1505.03068*, 2015.

[139] P. Festa, P. M. Pardalos, M. G. C. Resende, and C. C. Ribeiro. Randomized heuristics for the MAX-CUT problem. *Optimization methods and software*, 17(6):1033–1058, 2002.

[140] R. Martí, A. Duarte, M. Laguna. Advanced scatter search for the max-cut problem. *INFORMS Journal on Computing*, 21(1):26–38, 2009.

[141] S. Burer, R. Monteiro, Y. Zhang. Rank-two relaxation heuristics for max-cut and other binary quadratic programs. *SIAM Journal on Optimization*, 12(2):503–521, 2002.

[142] T. Wang, A.V. Karthik, B. Wu, J. Yao, and J. Roychowdhury. MAPP: The Berkeley Model and Algorithm Prototyping Platform. In *Proc. IEEE CICC*, pages 461–464, September 2015. DOI link.

[143] T. Wang, A. V. Karthik, B. Wu and J. Roychowdhury. MAPP: A platform for prototyping algorithms and models quickly and easily. In *IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO)*, pages 1–3. IEEE, 2015.

[144] T. Wang and C.C. McAndrew. A Generic Formalism to Model ESD Snapback for Robust Circuit Simulation. In *2018 40th Electrical Overstress/Electrostatic Discharge Symposium (EOS/ESD)*, pages 1–8. IEEE, 2018.

[145] T. Wang and J. Roychowdhury. Multiphysics Modelling and Simulation in Berkeley MAPP. In *Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO), 2016 IEEE MTT-S International Conference on*, pages 1–3. IEEE, 2016.

[146] T. Wang and J. Roychowdhury. Modelling Optical Devices and Systems in MAPP. *Technical Report No. UCB/EECS-2017-160*, 2017.

[147] T. Wang and J. Roychowdhury. Design Tools for Oscillator-based Computing Systems. In *Proc. IEEE DAC*, pages 188:1–188:6, 2015. DOI link.

[148] L.W. Nagel. *SPICE2: a computer program to simulate semiconductor circuits*. PhD thesis, EECS department, University of California, Berkeley, Electronics Research Laboratory, 1975. Memorandum no. ERL-M520.

[149] D. O. Pederson and A. Sangiovanni-Vincentelli. SPICE 3 Version 3F5 User's Manual. *Dept. EECS, Univ. California, Berkeley, CA*, 1994.

[150] P. Nenzi and H. Vogt. Ngspice Users Manual Version 26 (Describes ngspice-26 release version). 2014.

[151] A. Davis. The gnu circuit analysis package. 2006.

[152] M. E. Brinson and S. Jahn. Qucs: A GPL software package for circuit simulation, compact device modelling and circuit macromodelling from DC to RF and beyond. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, 22(4):297–319, 2009.

[153] E. R. Keiter, T. Mei, T. V. Russo, R. L. Schiek, H. K. Thornquist, J. C. Verley, D. A. Fixel, T. S. Coffey, R. P. Pawlowski, C. E. Warrender, et al. Xyce parallel electronic simulator users' guide, Version 6.0. 1. Technical report, Raytheon, Albuquerque, NM; Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), 2014.

[154] A. Vladimirescu. *The SPICE book*. John Wiley & Sons, Inc., 1994.

[155] E. Zhu. SUGAR 3.0: A MEMS Simulation Program (User's Guide). 2002.

[156] J. Wyatt, D. Mikulecky, and J. DeSimone. Network modelling of reaction-diffusion systems and their numerical solution using SPICE. *Chemical Engineering Science*, 35(10):2115–2127, 1980.

[157] R. Schiek, C. Warrender, C. Teeter, J. Aimone, H. Thornquist, T. Mei, and A. Duda. Simulating Neural Systems with Xyce. *No. SAND2012-10628. Sandia National Laboratories*, 2012.

[158] F. Liu and B. Hodges. Dynamic river network simulation at large scale. In *Proceedings of the 49th Annual Design Automation Conference*, pages 723–728. ACM, 2012.

[159] Accellera. Verilog-AMS Language Reference Manual, version 2.4, 2014.

[160] E. Kononov. *Modeling photonic links in Verilog-A*. PhD thesis, Massachusetts Institute of Technology, 2013.

[161] T. Wang, A. V. Karthik, B. Wu and J. Roychowdhury. Poster: MAPP: The Berkeley Model and Algorithm Prototyping Platform. In *IEEE International Conference on Software Engineering (ICSE)*, volume 2, pages 825–826. IEEE, 2015.

[162] D. Amsallem and J. Roychowdhury. ModSpec: An open, flexible specification framework for multi-domain device modelling. In *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, pages 367–374. IEEE, 2011.

[163] B. Wu and J. Roychowdhury. Efficient per-element distortion contribution analysis via harmonic balance adjoints. In *Proc. IEEE CICC*, pages 1–4, Sept 2014.

[164] MAPP: The Berkeley Model and Algorithm Prototyping Platform. Web site: https://github.com/jaijeet/MAPP.

[165] S. M. Sze and K. K. Ng. *Physics of semiconductor devices*. John Wiley and Sons, 2006.

[166] L. Lemaitre, G. Coram, C. C. McAndrew, and K. Kundert. Extensions to Verilog-A to support compact device modeling. In *Behavioral Modeling and Simulation, 2003. BMAS 2003. Proceedings of the 2003 International Workshop on*, pages 134–138. IEEE, 2003.

[167] R. Ida and C.C. McAndrew. A physically-based behavioral snapback model. In *Electrical Overstress/Electrostatic Discharge Symposium (EOS/ESD)*, pages 1–5. IEEE, 2012.

[168] L. O. Chua. Memristor-the missing circuit element. *IEEE Transactions on Circuit Theory*, 18(5):507–519, 1971.

[169] S.R. Williams. How We Found The Missing Memristor. *IEEE Spectrum*, 45(12):28–35, December 2008.

[170] K. Kundert. Hidden State in SpectreRF. *The Designer's Guide*, 2013.

[171] C. C. McAndrew, G. J. Coram, K. K. Gullapalli, J. R. Jones, L. W. Nagel, A.S. Roy, J. Roychowdhury, A.J. Scholten, Geert D.J. Smit, X. Wang and S. Yoshitomi. Best Practices for Compact Modeling in Verilog-A. *IEEE J. Electron Dev. Soc.*, 3(5):383–396, September 2015.

[172] L. Wei, C.E. Gill, W. Li, R. Wang and M. Zunino. A convergence robust method to model snapback for ESD simulation. In *CAS 2011 Proceedings (2011 International Semiconductor Conference)*, 2011.

[173] Z. Jiang, S. Yu, Y. Wu, J. H. Engel, X. Guan and H.-S. P. Wong. Verilog-A compact model for oxide-based resistive random access memory (RRAM). In *International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*, pages 41–44. IEEE, 2014.

[174] P.-Y. Chen and S. Yu. Compact Modeling of RRAM Devices and Its Applications in 1T1R and 1S1R Array Design. *IEEE Transactions on Electron Devices*, 62(12):4022–4028, 2015.

[175] G. Coram. How to (and how not to) write a compact model in Verilog-A. In *Behavioral Modeling and Simulation, 2004. BMAS 2004. Proceedings of the 2004 International Workshop on*, pages 97–106. IEEE, 2004.

[176] A.G. Mahmutoglu, T. Wang, A. Gupta and J. Roychowdhury. Well-posed Device Models for Electrical Circuit Simulation. https://nanohub.org/resources/26199, Mar 2017.

[177] T. Wang and J. Roychowdhury. Guidelines for Writing NEEDS-compatible Verilog-A Compact Models. https://nanohub.org/resources/18621, Jun 2013.

[178] S. Kvatinsky, E. G. Friedman, A. Kolodny and U. C. Weiser. TEAM: threshold adaptive memristor model. *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, 60(1):211–221, 2013.

[179] D. FitzPatrick and I. Miller. *Analog behavioral modeling with the Verilog-A language*. Springer Science & Business Media, 2007.

[180] Jacques Hadamard. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton university bulletin*, 13(49-52):28, 1902.

[181] C. Yakopcic, T. M. Taha, G. Subramanyam and R. E. Pino. Generalized memristive device SPICE model and its application in circuit design. *IEEE Trans. CAD*, 32(8):1201–1214, 2013.

[182] Sybil P Parker. *McGraw-Hill Dictionary of Scientific and Technical Terms*. McGraw-Hill Book Co., 1989.

[183] J. Roychowdhury and R. Melville. Delivering Global DC Convergence for Large Mixed-Signal Circuits via Homotopy/Continuation Methods. *IEEE Trans. CAD*, 25:66–78, Jan 2006.

[184] C.H. Diaz and S. Kang. New algorithms for circuit simulation of device breakdown. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11:1344–1354, 1992.

[185] C. Jiao and Z. Yu. A robust novel technique for SPICE simulation of ESD snapback characteristic. In *8th International Conference on Solid-State and Integrated Circuit Technology Proceedings*, pages 1367–1369. IEEE, 2006.

[186] Y. Zhou, D. Connerney, R. Carroll, and T. Luk. Modeling MOS snapback for circuit-level ESD simulation using BSIM3 and VBIC models. In *Sixth international symposium on quality electronic design*, pages 476–481. IEEE, 2005.

[187] J. Li, S. Joshi, R. Barnes, E. Rosenbaum. Compact modeling of on-chip ESD protection devices using Verilog-A. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1047–1063, 2006.

[188] Y. Zhou, J. Hajjar, A.W. Righter, and K.P. Lisiak. Modeling Snapback of LVTSCR Devices for ESD Circuit Simulation Using Advanced BJT and MOS Models. In *Electrical Overstress/Electrostatic Discharge Symposium (EOS/ESD)*, volume 29, page 175, 2007.

[189] R. Mertens, E. Rosenbaum. A physics-based compact model for SCR devices used in ESD protection circuits. In *IEEE International Reliability Physics Symposium (IRPS)*, pages 2B–2. IEEE, 2013.

[190] P. A. Juliano, E. Rosenbaum. A novel SCR macromodel for ESD circuit simulation. In *International Electron Devices Meeting*, pages 14–3. IEEE, 2001.

[191] J. McPherson, J. Kim-, A. Shanware and H. Mogul. Thermochemical description of dielectric breakdown in high dielectric constant materials. *Applied Physics Letters*, 82:2121, 2003.

[192] C. Yakopcic. *Memristor Device Modeling and Circuit Design for Read Out Integrated Circuits, Memory Architectures, and Neuromorphic Systems*. PhD thesis, University of Dayton, 2014.

[193] S. Kvatinsky, K. Talisveyberg, D. Fliter, E. G. Friedman, A. Kolodny and U. C. Weiser. Verilog-A for memristors models. In *CCIT Tech. Rep. 801*, 2011.

[194] A. Gupta, T. Wang, A. G. Mahmutoglu and J. Roychowdhury. STEAM: Spline-based Tables for Efficient and Accurate Device Modelling. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 463–468, Jan 2017.

[195] T. Wang and Z. Ye. Robust Passive Macro-model Generation with Local Compensation. *IEEE Trans. Microwave Theory Tech.*, 60(8):2313–2328, 2012.

[196] Z. Ye, T. Wang and Y. Li. Domain-Alternated Optimization for Passive Macromodeling. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(10):2244–2255, 2015.

[197] S. Skelboe. Computation of the periodic steady-state response of nonlinear networks by extrapolation methods. *IEEE Trans. Ckts. Syst.*, CAS-27:161–175, 1980.

[198] Pranav N Ashar. Implementation of algorithms for the periodic-steady-state analysis of nonlinear circuits. Technical report, 1989.

[199] P. Kinget. Private communication, June 1997.

[200] R. Adler. A study of locking phenomena in oscillators. *Proceedings of the I.R.E. and Waves and Electrons*, 34:351–357, June 1946.

[201] A. H. Taub, editor. *John von Neumann: Collected Works: Volume I: Logic, Theory of Sets and Quantum Mechanics*. Pergamon Press, New York, NY, USA, 1961. See also volumes II–VI [71, 202–205].

[202] A. H. Taub, editor. *John von Neumann: Collected Works. Volume II: Operators, Ergodic Theory and Almost Periodic Functions in a Group*. Pergamon Press, New York, NY, USA, 1961. See also volumes I, III–VI [71, 201, 203–205].

[203] A. H. Taub, editor. *John von Neumann: Collected Works. Volume III: Rings of Operators*. Pergamon Press, New York, NY, USA, 1961–1963. See also volumes I–II, IV–VI [71, 201, 202, 204, 205].

[204] A. H. Taub, editor. *John von Neumann: Collected Works. Volume IV: Continuous Geometry and Other Topics*. Pergamon Press, New York, NY, USA, 1962. See also volumes I–III, V–VI [71, 201–203, 205].

[205] A. H. Taub, editor. *John von Neumann: Collected Works. Volume VI: Theory of Games, Astrophysics, Hydrodynamics and Meteorology*. Pergamon Press, New York, NY, USA, 1963. See also volumes I–V [71, 201–204].