30 January 2026

QFlux: Quantum Circuit Implementations of Molecular Dynamics. Part III - State Initialization and Unitary Decomposition

Alexander V Soudackov[1], Delmar G A Cabral[1], Brandon C Allen[1], Xiaohan Dan[1], Nam P Vu[1,2,3], Cameron Cianci[4], Rishab Dutta[1], Sabre Kais[5], Eitan Geva[6], Victor S Batista[1,7]

1. Department of Chemistry Yale University
2. Department of Electrical Engineering and Computer Science Massachusetts Institute of Technology
3. Research Laboratory of Electronics Massachusetts Institute of Technology
4. Department of Physics University of Connecticut
5. Department of Electrical and Computer Engineering North Carolina State University
6. Department of Chemistry University of Michigan
7. Yale Quantum Institute Yale University

Abstract

This tutorial builds upon the foundations established in Part~II to present a unified, implementation-oriented overview of quantum state initialization and unitary decomposition for n-qubit systems, available in QFlux. We begin with the preparation of arbitrary quantum states and develop two complementary constructions: (i) a recursive multiplexor method (after Shende et al.) that disentangles qubits via multiplexed Ry and Rz rotations, and (ii) an algebraic scheme based on uniformly controlled rotations (UCRs) (after Möttönen et al.) that realizes the same mapping through analytically defined rotation networks with predictable gate counts. We then extend these state-preparation tools to generic unitary synthesis through Givens-rotation, column-by-column, and recursive cosine-sine (CSD) and quantum Shannon (QSD) decompositions, explicitly linking linear-algebraic factorizations to executable circuits while tracking CNOT complexity. Finally, we introduce the Walsh decomposition for diagonal unitaries and show how Gray-code ordering and local CNOT cancellations yield an O(2^n) entangling-gate cost, providing shallow, NISQ-friendly implementations. Taken together, these techniques form a pedagogical bridge from matrix analysis to hardware-efficient circuit constructions, offering clear design rules, closed-form parameters, and scalable synthesis pathways for simulation and experiment.

QFlux: Quantum Circuit Implementations of Molecular Dynamics. Part III – State Initialization and Unitary Decomposition

Alexander V. Soudackov,[†] Delmar G. A. Cabral,[†] Brandon C. Allen,[†] Xiaohan Dan,[†] Nam P. Vu,[†,‡,¶] Cameron Cianci,[§] Rishab Dutta,[†] Sabre Kais,[‖] Eitan Geva,[⊥] and Victor S. Batista[*,†,#]

[†]*Department of Chemistry, Yale University, New Haven, CT 06520, USA*

[‡]*Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, USA*

[¶]*Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA*

[§]*Department of Physics, University of Connecticut, Storrs, CT 06268, USA*

[‖]*Department of Electrical and Computer Engineering, Department of Chemistry, North Carolina State University, Raleigh, North Carolina 27606, USA*

[⊥]*Department of Chemistry, University of Michigan, Ann Arbor, MI 48109, USA*

[#]*Yale Quantum Institute, Yale University, New Haven, CT 06511, USA*

E-mail: victor.batista@yale.edu

Abstract

This tutorial builds upon the foundations established in Part II to present a unified, implementation-oriented overview of quantum state initialization and unitary decomposition for n-qubit systems, available in **QFlux**. We begin with the preparation of arbitrary quantum states and develop two complementary constructions: (i) a recursive multiplexor method (after Shende *et al.*) that disentangles qubits via multiplexed R_y and R_z rotations, and (ii) an algebraic scheme based on uniformly controlled rotations (UCRs) (after Möttönen *et al.*) that realizes the same mapping through analytically defined rotation networks with predictable gate counts. We then extend these state-preparation tools to generic unitary synthesis through Givens-rotation, column-by-column, and recursive cosine–sine (CSD) and quantum Shannon (QSD) decompositions, explicitly linking linear-algebraic factorizations to executable circuits while tracking CNOT complexity. Finally, we introduce the Walsh decomposition for diagonal unitaries and show how Gray-code ordering and local CNOT cancellations yield an $O(2^n)$ entangling-gate cost, providing shallow, NISQ-friendly implementations. Taken together, these techniques form a pedagogical bridge from matrix analysis to hardware-efficient circuit constructions, offering clear design rules, closed-form parameters, and scalable synthesis pathways for simulation and experiment.

1 Introduction

Quantum computing offers a powerful framework for simulating and manipulating quantum systems by operating directly within the Hilbert space of many-body states. A prerequisite for essentially all quantum algorithms is the ability to efficiently *initialize quantum states* and *decompose unitary operators* into executable gate sequences. These two tasks-state preparation and unitary synthesis-form the backbone of quantum simulation, quantum chemistry, linear-algebraic algorithms, machine learning, and variational methods. Despite their fundamental role, translating abstract mathematical objects such as complex vectors and matrices into compact, hardware-compatible quantum circuits remains one of the central technical challenges in practical quantum computing.

This paper presents **Part III** of the QFlux[1] tutorial series and focuses on the constructive methods that make quantum simulations executable. Building on the physical intuition and circuit-level propagation strategies developed in **Parts I and II**, this installment addresses how quantum states and operators are systematically mapped onto gate-based architectures. The emphasis is on implementation-oriented algorithms that connect linear-algebraic factorizations to explicit circuit constructions with predictable resource requirements.

While the individual building blocks discussed here-state preparation, unitary synthesis, and diagonal-unitary decomposition-are well established in the quantum-computing literature,[2–6] the contribution of this tutorial lies in their unified, implementation-level treatment within a single coherent framework. Rather than presenting these techniques in isolation, we integrate arbitrary state initialization, generic unitary decomposition, and Walsh-based synthesis of diagonal operators[7–11] into a consistent end-to-end pipeline that maps abstract vectors and matrices directly to executable quantum circuits. Throughout, gate counts-particularly CNOT complexity-are tracked systematically and transparently, enabling direct comparison among synthesis strategies and informed method selection for near-term devices. All constructions are implemented natively within QFlux, providing a reproducible, software-integrated workflow that carries the reader from linear-algebraic input (state vectors or operators) to hardware-ready circuits.

We begin by introducing algorithms for state preparation on an n-qubit register. Two complementary approaches are presented. The first is a *recursive multiplexor method*,[2] which constructs arbitrary quantum states through hierarchically controlled R_y and R_z rotations. The second is a *uniformly controlled rotation* (UCR) scheme,[3] which encodes amplitudes and phases analytically using structured rotation networks. Both approaches are illustrated with explicit circuit examples and analyzed in terms of gate counts and CNOT scaling.

The second part of the tutorial extends these ideas to the decomposition of arbitrary unitary matrices. We introduce several widely used synthesis strategies, including Givens-rotation-based QR decomposition,[4] column-by-column construction with correction gates,[6] and recursive factorizations such as the *Cosine-Sine Decomposition (CSD)*[5] and the *Quantum Shannon Decomposition (QSD)*.[2] These methods progressively reduce an n-qubit unitary into products of simpler controlled operations that can be implemented using a universal gate set.

Finally, we present the *Walsh decomposition* for diagonal unitaries. By expanding diagonal Hamiltonians in the Walsh-Fourier basis[7,8] and exploiting Gray-code ordering,[9,12] this approach yields circuits with only $O(2^n)$ entangling gates,[9] significantly reducing circuit depth and improving fidelity on noisy intermediate-scale quantum (NISQ) devices.[10] Because diagonal unitaries arise ubiquitously in Hamiltonian simulation and time evolution, this method plays a recurring role throughout the QFlux series.

Throughout the tutorial, theory and implementation are tightly integrated. Code examples accompany each algorithm to demonstrate the complete workflow-from deriving matrix factorizations to mapping them onto gate sequences and optimizing the resulting circuits. Together, these techniques provide a coherent bridge from abstract unitary design to hardware-efficient circuit realization.

Within the broader tutorial sequence, **Part III** establishes the circuit-synthesis infrastructure required for all subsequent developments. The tools introduced here are used directly in **Part IV**, which extends QFlux to open quantum systems through Lindblad dynamics and dilation techniques,[10,13–18] in **Part V**, which introduces adaptive variational algorithms for noisy quantum hardware, and in **Part VI**, which addresses non-Markovian dynamics via generalized quantum master equations.[18] These interdependencies are summarized schematically in Fig. 1, which provides a visual roadmap for the synthesis pipeline developed in this work.

As summarized in Fig. 1, synthesis strategy in QFlux is guided by operator structure: diagonal or phase-only operators are implemented most efficiently using Walsh-based decom-

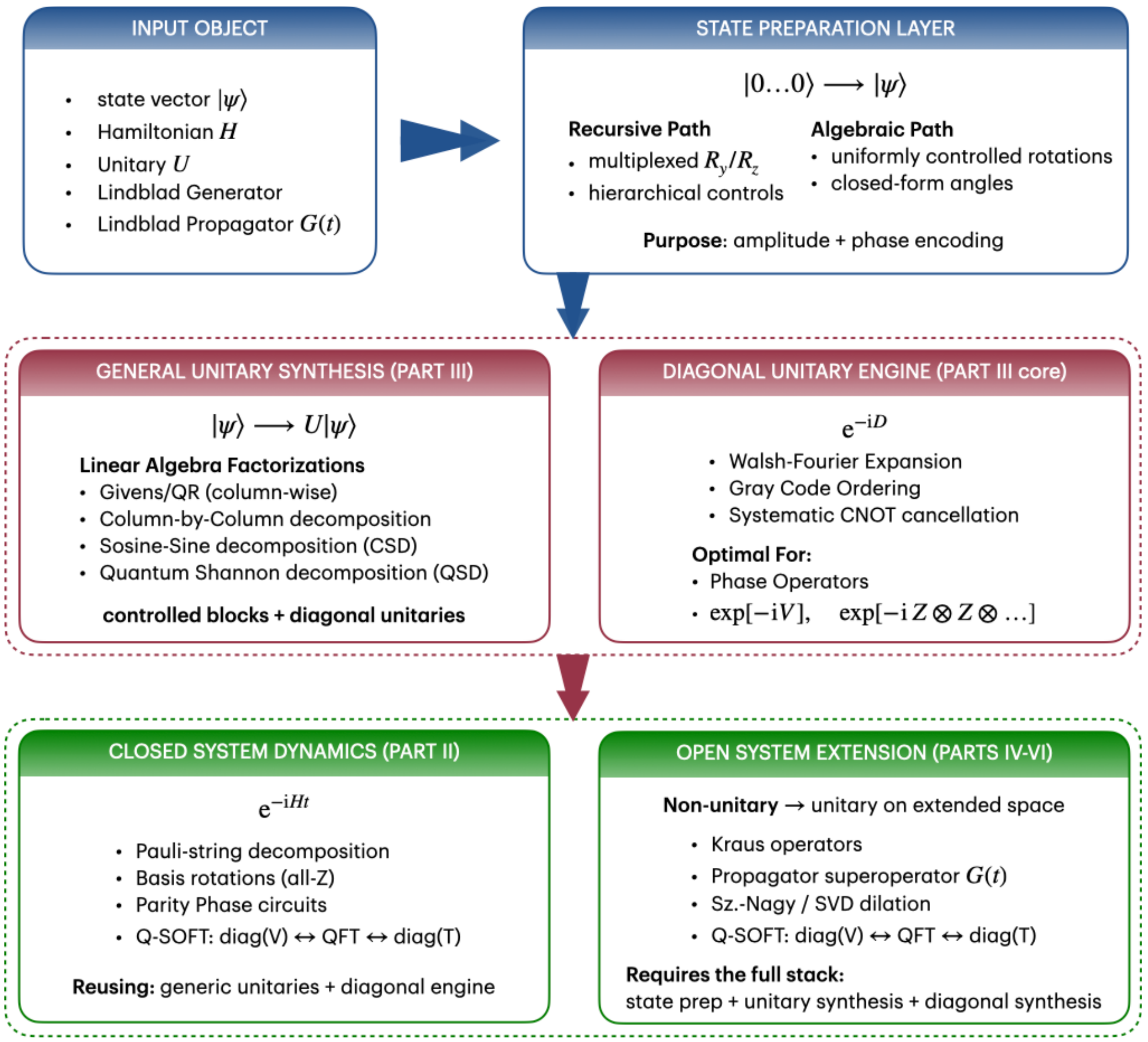

INPUT OBJECT
state vector $|\psi\rangle$
Hamiltonian H
Unitary U
Lindblad Generator
Lindblad Propagator $G(t)$
STATE PREPARATION LAYER
$|0\ldots0\rangle \longrightarrow |\psi\rangle$
Recursive Path
multiplexed R_y/R_z
hierarchical controls
Algebraic Path
uniformly controlled rotations
closed-form angles
Purpose: amplitude + phase encoding
GENERAL UNITARY SYNTHESIS (PART III)
$|\psi\rangle \longrightarrow U|\psi\rangle$
Linear Algebra Factorizations
Givens/QR (column-wise)
Column-by-Column decomposition
Sosine-Sine decomposition (CSD)
Quantum Shannon decomposition (QSD)
controlled blocks + diagonal unitaries
DIAGONAL UNITARY ENGINE (PART III core)
e^{-iD}
Walsh-Fourier Expansion
Gray Code Ordering
Systematic CNOT cancellation
Optimal For:
Phase Operators
$\exp[-iV]$, $\exp[-i Z\otimes Z\otimes \ldots]$
CLOSED SYSTEM DYNAMICS (PART II)
e^{-iHt}
Pauli-string decomposition
Basis rotations (all-Z)
Parity Phase circuits
Q-SOFT: diag(V) ↔ QFT ↔ diag(T)
Reusing: generic unitaries + diagonal engine
OPEN SYSTEM EXTENSION (PARTS IV-VI)
Non-unitary → unitary on extended space
Kraus operators
Propagator superoperator $G(t)$
Sz.-Nagy / SVD dilation
Q-SOFT: diag(V) ↔ QFT ↔ diag(T)
Requires the full stack:
state prep + unitary synthesis + diagonal synthesis

Figure 1: **Unified circuit-synthesis workflow underlying QFlux (Parts II–VI).** Abstract inputs-state vectors, generic unitaries, Hamiltonians, and open-system maps-are translated into executable quantum circuits through a layered synthesis stack. Arbitrary state preparation is achieved using multiplexor or uniformly controlled rotation (UCR) networks. Generic unitaries are constructed via linear-algebraic factorizations (Givens/QR, column-by-column methods, and CSD/QSD), yielding controlled blocks and diagonal sub-units. Diagonal operators are implemented efficiently using Walsh decompositions with Gray-code ordering, enabling CNOT cancellation and shallow circuits. Closed-system time evolution e^{-iHt} (Part II) and open-system simulations via dilation (Part IV) both reuse this common synthesis infrastructure, highlighting a single, modular pathway from abstract operators to hardware-ready quantum circuits.

position;[9] generic unitaries on small registers are naturally handled via column-by-column decomposition;[6] larger generic unitaries benefit from recursive cosine–sine or quantum Shannon decompositions (CSD/QSD);[2,5] and state preparation or debugging tasks favor recursive multiplexor constructions[2] that prioritize transparency over minimal gate count.

Relation to modern compiler-level synthesis. In practical quantum computing workflows, many users rely on automated compiler pipelines rather than explicit, hand-crafted circuit decompositions. For small systems-most notably one- and two-qubit blocks-optimal or near-optimal synthesis is often achieved using Cartan (KAK) decompositions, which factor generic SU(4) unitaries into a minimal entangling core supplemented by local rotations and form the backbone of many modern transpilers.[2,19,20] At larger scales, contemporary compilation strategies increasingly emphasize hardware-aware transpilation, qubit routing, and gate resynthesis, as well as variational or numerical compilation methods that optimize parameterized circuits directly against a target unitary under device-specific constraints.[21–24] While such approaches can substantially reduce circuit depth or improve fidelity on specific hardware platforms, they typically operate as black-box optimizers and provide limited analytic transparency. The constructions presented in this work therefore serve a complementary role: they provide deterministic, structurally explicit mappings from quantum states and operators to executable circuits, which are essential for custom or highly structured unitaries, Hamiltonian simulation, algorithmic validation, and situations where analytic control over circuit form, parameterization, or scaling is required. In this sense, the QFlux synthesis framework is not intended to replace modern compilers, but rather to supply a transparent and reproducible foundation that informs, constrains, and seeds higher-level compilation and optimization workflows.

2 Method Selection

This section elaborates on the practical tradeoffs underlying the method-selection guidelines summarized in Fig. 1. The appropriate choice of synthesis method depends on both the structure of the target operator and the constraints imposed by available quantum hardware.

For arbitrary state preparation, two complementary strategies are emphasized. Uniformly controlled rotations (UCRs)[3] are often preferred when control patterns are regular or when a modular, layered circuit structure is desirable. They achieve asymptotic scaling comparable to recursive multiplexor constructions while offering greater circuit regularity and improved compiler friendliness. Recursive multiplexors, by contrast, provide fully general and exact constructions with maximal conceptual transparency,[2] but typically incur higher CNOT overhead, which can limit their direct applicability on near-term devices.

When the target operator has diagonal or phase-only structure, Walsh-based synthesis is optimal. By exploiting the commutativity of diagonal terms together with Gray-code ordering,[9,12] this approach achieves near-minimal two-qubit gate counts and is particularly well suited to NISQ-era hardware. In contrast, cosine–sine and quantum Shannon decompositions (CSD/QSD) are asymptotically optimal for generic unitaries,[2,5] but are primarily of theoretical or compiler-level interest due to their circuit depth and control complexity.

Practical method selection is ultimately constrained by hardware-specific considerations that lie beyond the scope of this introductory tutorial. Limited qubit connectivity, native gate sets, and compilation- and noise-induced errors can substantially alter the effective cost of otherwise optimal decompositions. On near-term devices, synthesis strategies that minimize two-qubit gate counts and exploit structural features of the target operator are therefore likely to outperform formally optimal but deeper constructions.

To guide method selection in the context of state preparation, Table 1 summarizes the key practical tradeoffs between recursive multiplexor-based constructions and uniformly controlled rotations.

Despite their higher CNOT cost, multiplexor constructions remain valuable when fine-grained control, interpretability, or systematic debugging of state-preparation circuits is required. In most practical settings, uniformly controlled rotations serve as the default choice for arbitrary state preparation due to their regular structure and compiler-friendly implementation.

Table 1: Comparison of multiplexor-based and uniformly controlled rotation (UCR) state-preparation methods.

Feature	**Multiplexors**	**UCRs**
CNOT count	$2^{n+1} - 2n$	$2^{n+2} - 4n - 4$
1Q rotations	$O(2^n)$	$2^{n+2} - 5$
Structure	Recursive	Regular
Parameter form	Numerical	Analytic
Compiler support	Moderate	High
Typical use	Transparency, debugging,small-n	Scalable synthesis,automation

3 State Initialization Algorithms

State initialization is a fundamental operation in quantum computing, as it defines how a quantum register of n qubits is prepared in a specified target state. Starting from the vacuum state

$$|00\cdots 0\rangle = |0\rangle^{\otimes n} = \underbrace{\begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} \otimes \cdots \otimes \begin{pmatrix}1\\0\end{pmatrix}}_{n \text{ qubits}} = (1, 0, \ldots, 0)^\top. \tag{3.1}$$

the objective is to construct a quantum circuit that transforms it into an arbitrary target state

$$|\psi\rangle = (c_1, c_2, \ldots, c_{2^n})^\top, \tag{3.2}$$

with complex amplitudes c_j satisfying $\sum_j |c_j|^2 = 1$.

Roadmap. Sections 3.1 and 3.2 introduce two complementary strategies for preparing arbitrary quantum states-recursive and algebraic-whose circuit primitives are reused throughout the unitary-synthesis methods developed in Section 4.

We focus on two state-preparation algorithms implemented within QFlux. The first is a recursive, multiplexor-based construction following Shende *et al.*,[2] which hierarchically disentangles qubits using controlled R_y and R_z rotations. The second, due to Möttönen *et al.*,[3] employs uniformly controlled rotations (UCRs) to encode amplitudes and phases via an explicit algebraic mapping with predictable resource requirements. Both methods deterministically prepare arbitrary quantum states from the vacuum and serve as foundational building blocks for the unitary-synthesis techniques developed in later sections.

In terms of gate complexity, recursive demultiplexing for an n-qubit register requires $2^{n+1} - 2n$ CNOT gates,[2] whereas the UCR-based approach uses $2^{n+2} - 4n - 4$ CNOT gates and $2^{n+2} - 5$ single-qubit rotations. Together, these constructions provide a practical and scalable foundation for modern quantum state preparation, underpinning applications in quantum simulation, optimization, and data encoding.

3.1 Quantum Multiplexors

Goal of this subsection. We show how an arbitrary n-qubit quantum state can be prepared from the vacuum by recursively disentangling qubits using multiplexed single-qubit rotations, reducing the problem dimension by one qubit at each step.

A *quantum multiplexor* is a gate that applies different single-qubit operations depending on the state of a set of control qubits. Multiplexors play a central role in efficient quantum state preparation and decomposition algorithms. To understand them, we begin from the most fundamental idea: how to “disentangle" one qubit from a general n-qubit state.

Step 1: Decomposing the Target Quantum State

Consider an arbitrary n-qubit state $|\psi\rangle$ expressed in the computational basis:

$$|\psi\rangle = \sum_{k_1,k_2,\ldots,k_n \in \{0,1\}} c_{k_1 k_2 \ldots k_n} |k_1 k_2 \ldots k_n\rangle \,. \tag{3.3}$$

Here, each binary string $(k_1 k_2 \ldots k_n)$ labels a computational basis state, and each coefficient $c_{k_1 k_2 \ldots k_n}$ is a complex amplitude. To isolate the last qubit, we group terms by the first $n-1$

bits:

$$|\psi\rangle = \sum_{k_1,k_2,\ldots,k_{n-1}\in\{0,1\}} |k_1k_2\ldots k_{n-1}\rangle \otimes \left[c_{k_1k_2\ldots k_{n-1}0}\,|0\rangle + c_{k_1k_2\ldots k_{n-1}1}\,|1\rangle\right]. \tag{3.4}$$

The bracketed term defines a single-qubit state, which we denote:

$$\left|\rho_{k_1k_2\ldots k_{n-1}}\right\rangle = c_{k_1k_2\ldots k_{n-1}0}\,|0\rangle + c_{k_1k_2\ldots k_{n-1}1}\,|1\rangle\,. \tag{3.5}$$

Thus, $|\psi\rangle$ can be viewed as a superposition of $(n-1)$-qubit states, each entangled with one single-qubit state $\left|\rho_{k_1k_2\ldots k_{n-1}}\right\rangle$.

Intuition. Grouping amplitudes by the last qubit isolates a conditional single-qubit state for each configuration of the remaining qubits, making it possible to remove entanglement one qubit at a time using controlled rotations.

Step 2: Aligning the Last Qubit on the Bloch Sphere

Each $\left|\rho_{k_1k_2\ldots k_{n-1}}\right\rangle$ can be visualized as a point on the Bloch sphere. Our goal is to rotate this qubit so that it aligns with the north pole, $|0\rangle$, while leaving the other qubits untouched. This is achieved through a pair of single-qubit rotations:

$$R_y(-\theta_{k_1k_2\ldots k_{n-1}})\,R_z(-\varphi_{k_1k_2\ldots k_{n-1}})\left|\rho_{k_1k_2\ldots k_{n-1}}\right\rangle = r_{k_1k_2\ldots k_{n-1}}\mathrm{e}^{\mathrm{i}t_{k_1k_2\ldots k_{n-1}}}\,|0\rangle\,. \tag{3.6}$$

The angles $\varphi_{k_1k_2\ldots k_{n-1}}$ and $\theta_{k_1k_2\ldots k_{n-1}}$ are directly computed from the amplitudes $c_{k_1k_2\ldots k_{n-1}0}$ and $c_{k_1k_2\ldots k_{n-1}1}$. Geometrically, this corresponds to rotating the Bloch vector into the z-axis.

Intuition. Each conditional single-qubit state can be rotated to the north pole of the Bloch sphere, so a single pair of rotations suffices to erase the last qubit's entanglement for a fixed control configuration.

Step 3: Defining the Quantum Multiplexor

Because the rotation angles depend on the configuration of the control qubits $(k_1, \ldots, k_{n-1})$, the combined operation can be written as a block-diagonal matrix,

$$R^{(n)} = \bigoplus_{k_1,\ldots,k_{n-1}\in\{0,1\}} R_y(-\theta_{k_1k_2\ldots k_{n-1}})R_z(-\varphi_{k_1k_2\ldots k_{n-1}}), \tag{3.7}$$

that is, a direct sum of 2^{n-1} conditional single-qubit rotations acting on the last qubit. This operation is referred to as a **quantum multiplexor**. Acting on the state $|\psi\rangle$, it disentangles the final qubit,

$$R^{(n)} |\psi\rangle = \left|\psi^{(n-1)}\right\rangle \otimes |0\rangle \,, \tag{3.8}$$

where $\left|\psi^{(n-1)}\right\rangle$ denotes the reduced state of the remaining $n-1$ qubits. In this way, the multiplexor reduces the effective problem size from 2^n to 2^{n-1} amplitudes.

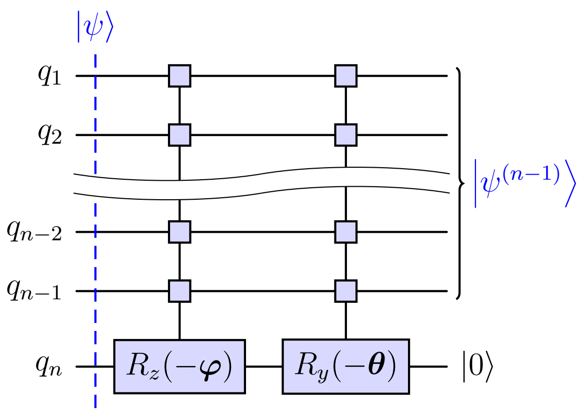

$|\psi\rangle$
q_1
q_2
q_{n-2}
q_{n-1}
q_n
$R_z(-\boldsymbol{\varphi})$
$R_y(-\boldsymbol{\theta})$
$|0\rangle$
$\left|\psi^{(n-1)}\right\rangle$

Figure 2: Quantum circuit implementing multiplexed $R_z^{(n)}$ and $R_y^{(n)}$ rotations, with angles conditioned on the binary state of the $n-1$ most significant qubits.

Intuition. The multiplexor applies all required conditional rotations in parallel, disentangling the last qubit and reducing the n-qubit problem to an $(n-1)$-qubit one, thereby enabling a recursive construction.

Step 4: Recursive Disentanglement

The same procedure can be applied recursively. After disentangling the last qubit, the multiplexor construction is repeated on the remaining $n-1$ qubits. After n iterations, all qubits are aligned to $|0\rangle$, yielding

$$\prod_{m=1}^{n}\left[R^{(m)}\otimes I^{\otimes(n-m)}\right]|\psi\rangle = \mathrm{e}^{\mathrm{i}\Phi}|00\ldots0\rangle\,, \tag{3.9}$$

up to a global phase $\mathrm{e}^{\mathrm{i}\Phi}$. To *prepare* the state $|\psi\rangle$, this procedure is simply reversed,

$$|\psi\rangle = \prod_{m=1}^{n}\left[\left(R^{(n-m+1)}\right)^{\dagger}\otimes I^{\otimes(m-1)}\right]D\,|00\ldots0\rangle\,, \tag{3.10}$$

where $D=\mathrm{e}^{-\mathrm{i}\Phi}$ removes the accumulated global phase.

3.1.1 Supporting Functions for State Preparation

Goal of this subsection. We outline the numerical routines and matrix-building primitives required to implement the recursive multiplexor state-preparation algorithm in practice.

1. Computing Bloch-Sphere Angles. Script S.1.1 shows how to compute the Bloch-sphere angles ϕ and θ corresponding to the single-qubit state $|\rho\rangle = c_0|0\rangle + c_1|1\rangle$, obtained according to the following rotation of the vacuum state:

$$|\rho\rangle = R_z(\phi)R_y(\theta)\,|0\rangle\,. \tag{3.11}$$

2. Rotation Matrices. The standard $SU(2)$ operators, corresponding to the $R_Z(\phi)$ and $R_Y(\theta)$ rotations used in all decompositions are provided in Script S.1.2.

3. Multiplexor Construction. Script S.1.3 builds the block-diagonal matrix introduced by Eq. (3.7).

4. Recursive Transformation. Script S.1.4 recursively applies multiplexors to map $|\psi\rangle$ to $|00...0\rangle$.

3.1.2 Example: Three-Qubit State Preparation

Script S.1.5 provides a simple example of a three-qubit state preparation.

3.1.3 Example: Coherent Wavepacket State Preparation on a 6-Qubit System

Script S.1.6 provides another example of a state preparation. In this example, the initial state describes a coherent wavepacket for a particle with mass $m = 1$ a.u. and frequency $\omega = 1$ a.u. in the position grid representation with $2^6 = 64$ grid points on the interval between -5 Bohr and 5 Bohr.

3.1.4 Quantum Circuit Implementation

The full quantum state preparation circuit based on multiplexed rotations is shown in Fig. 3. This circuit generates the target quantum state $|\psi\rangle$ from the initial vacuum state $|00\ldots0\rangle$. The operations $R_y^{(m)}$ and $R_z^{(m)}$ denote multiplexed rotations acting on qubit m with $m-1$ control qubits, and D represents a diagonal unitary operator that encodes the relative phases of the target amplitudes.

To execute this circuit on a physical quantum device, both the diagonal operator D and the multiplexed rotations R_z and R_y must be decomposed into elementary gates. The simplest example of such a decomposition is the two-qubit multiplexor,

$$U = U_1 \oplus U_2, \tag{3.12}$$

where U_1 and U_2 are single-qubit unitaries acting on the least significant qubit q_2, conditioned on the state of the control qubit q_1. Specifically, U_1 acts when $|q_1\rangle = |0\rangle$ and U_2 acts when

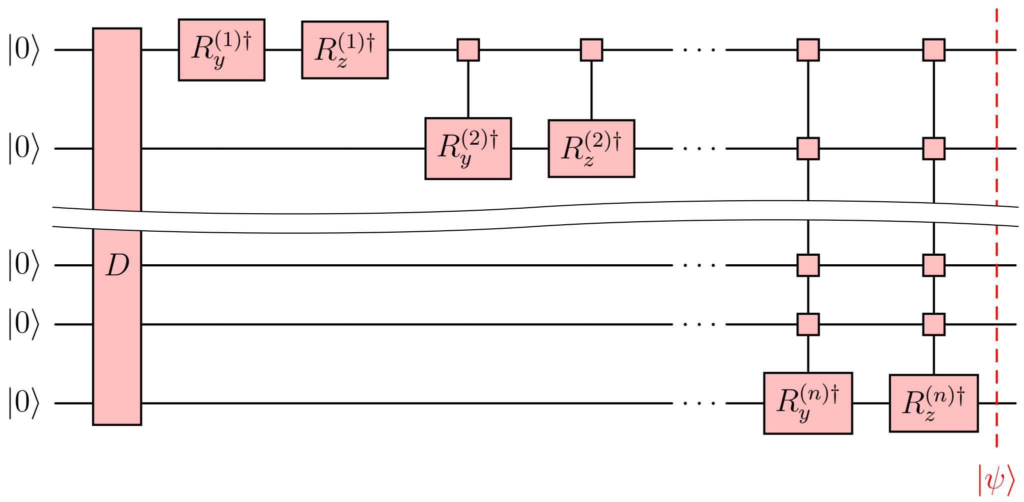


Figure 3: Quantum circuit for state preparation via multiplexed rotations. The operations $R_y^{(m)}$ and $R_z^{(m)}$ perform multiplexed rotations on qubit m with $m-1$ control qubits, while D is a diagonal operator encoding phase information.

$|q_1\rangle = |1\rangle$. For a general superposition $|q_1\rangle = c_0|0\rangle + c_1|1\rangle$, the multiplexor behaves as

$$(U_1 \oplus U_2)\,|q_1\rangle \otimes |q_2\rangle = c_0\,|0\rangle \otimes U_1\,|q_2\rangle + c_1\,|1\rangle \otimes U_2\,|q_2\rangle\,. \tag{3.13}$$

This effect can be realized by first applying U_1 to q_2 unconditionally, followed by a controlled operation $A = U_2U_1^\dagger$ conditioned on $|q_1\rangle = |1\rangle$. The unitary A can be expressed using the standard ZYZ decomposition:[25,26]

$$A = \mathrm{e}^{\mathrm{i}\Phi} R_z(\alpha) R_y(\beta) R_z(\gamma). \tag{3.14}$$

The corresponding quantum circuit for the two-qubit multiplexor is shown in Fig. 4.

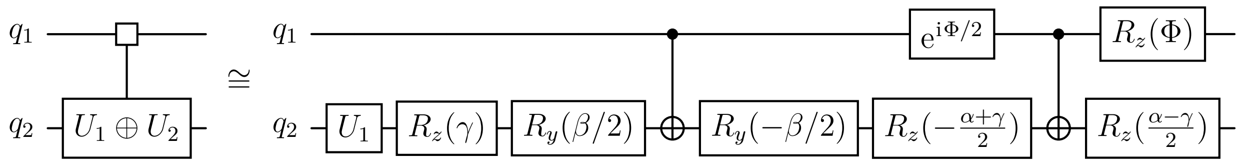


Figure 4: Decomposition of a general two-qubit multiplexor into elementary gates.

When U_1 and U_2 are single-axis rotation gates, $U_1 = R_\alpha(\theta_1)$ and $U_2 = R_\alpha(\theta_2)$, the circuit simplifies considerably, as shown in Fig. 5. Here, the rotation axis α can be either y or z.

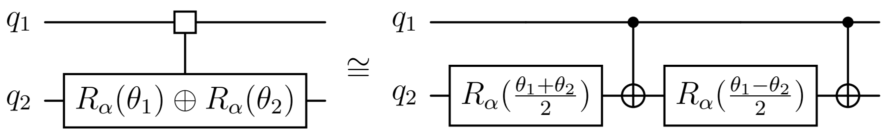
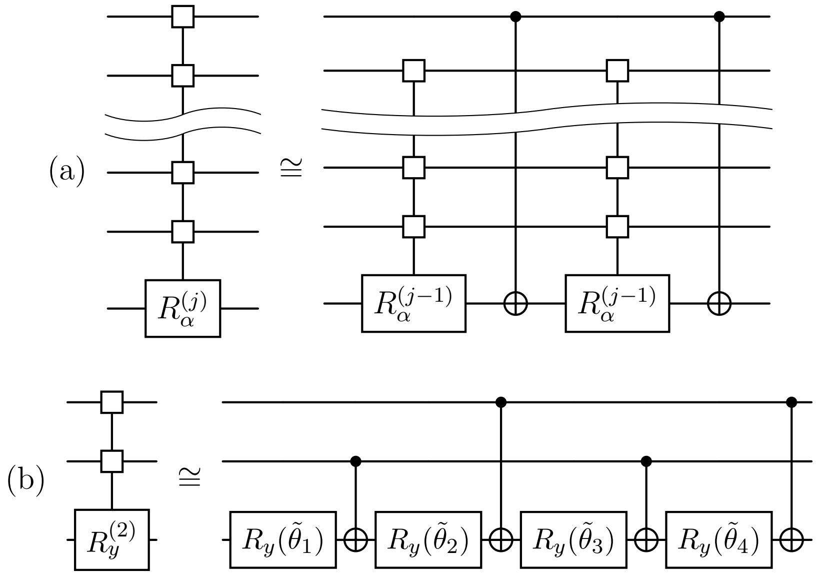

q_1
q_2
$R_\alpha(\theta_1) \oplus R_\alpha(\theta_2)$
$\cong$
q_1
q_2
$R_\alpha(\frac{\theta_1+\theta_2}{2})$
$R_\alpha(\frac{\theta_1-\theta_2}{2})$

Figure 5: Simplified decomposition of a two-qubit multiplexor $R_\alpha(\theta_1) \oplus R_\alpha(\theta_2)$, with $\alpha = y$ or z.

For an n-qubit register, a multiplexed rotation gate $R_\alpha^{(k)}$ with k control qubits can be recursively demultiplexed using 2^k CNOT gates.[2,27] The recursive pattern is illustrated in Fig. 6, showing how higher-level multiplexors reduce to smaller blocks. Overall, the complete state-preparation procedure requires $2^{n+1} - 2n$ CNOT gates,[2] providing a compact and systematic synthesis strategy.



$$\tilde{\theta}_1 = \frac{1}{4}\left(\theta_{00} + \theta_{01} + \theta_{10} + \theta_{11}\right), \quad \tilde{\theta}_2 = \frac{1}{4}\left(\theta_{00} - \theta_{01} + \theta_{10} - \theta_{11}\right),$$
$$\tilde{\theta}_3 = \frac{1}{4}\left(\theta_{00} - \theta_{01} - \theta_{10} + \theta_{11}\right), \quad \tilde{\theta}_4 = \frac{1}{4}\left(\theta_{00} + \theta_{01} - \theta_{10} - \theta_{11}\right).$$

Figure 6: Demultiplexing of multiplexed rotation gates. (a) Single recursive step for $R_\alpha^{(j)}$ with j control qubits. (b) Complete decomposition of $R_y^{(3)}$ showing how the four original rotation angles combine into new effective angles $\tilde{\theta}_i$.

Finally, the diagonal unitary D appearing in Eq. (3.10) can also be recursively decomposed using alternating layers of $R_z(\varphi)$ rotations and CNOT gates. As shown in Fig. 7, this construction requires only $2^{n+1} - 3$ CNOTs,[28] providing an efficient and compact implementation for phase encoding. This decomposition framework ensures that both the amplitude

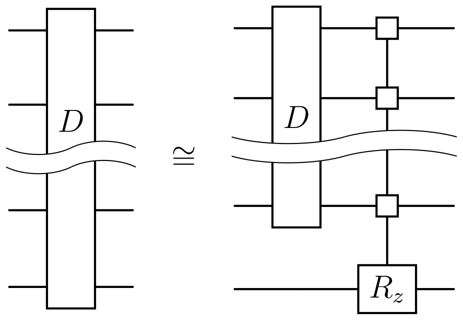


Figure 7: First step in the recursive decomposition of the diagonal unitary D. The structure alternates between R_z rotations and CNOT gates, yielding an efficient implementation with $2^{n+1} - 3$ CNOTs.

and phase components of the target quantum state are efficiently encoded using a minimal number of gates, making the multiplexed-rotation approach highly suitable for scalable state preparation on near-term quantum devices.

Checkpoint Summary: Recursive Multiplexor State Preparation

What problem did we solve? We constructed a recursive procedure to prepare an arbitrary n-qubit quantum state from the vacuum by successively disentangling qubits using multiplexed single-qubit rotations.

When should I use this method? Use this approach when conceptual transparency and step-by-step control over amplitude and phase encoding are important, or when validating/debugging state-preparation pipelines.

What is the asymptotic cost? CNOT gates: $2^{n+1} - 2n$

Single-qubit rotations: $O(2^n)$

3.2 Uniformly Controlled Rotations

While the recursive multiplexor construction of Section 2.1 offers a transparent and systematic route to arbitrary state preparation, it relies on hierarchical disentanglement steps whose circuit structure is inherently recursive. In this section, we present an alternative formulation based on *uniformly controlled rotations*,[3] which realizes the same mapping from the vacuum state $|0\ldots0\rangle$ to an arbitrary n-qubit target state $|\psi\rangle$ using an explicit, algebraic construction. In contrast to recursive synthesis, this approach encodes amplitudes and relative phases directly into structured rotation networks with analytically determined angles and predictable resource requirements. As a result, arbitrary state preparation can be achieved using only $2^{n+2}-4n-4$ CNOT gates and $2^{n+2}-5$ single-qubit rotations, yielding a compact, compiler-friendly circuit architecture well suited for scalable implementations and automated synthesis pipelines.

Definition

A **uniformly controlled rotation**, denoted $F_m^k(\mathbf{a},\boldsymbol{\alpha})$, acts on k *control qubits* and one *target qubit* m. It performs a sequence of 2^k single-qubit rotations about an axis $\mathbf{a} = (a_x, a_y, a_z)$, where each rotation angle α_j corresponds to one binary configuration of the control qubits.

Each controlled operation applies the rotation

$$R_{\mathbf{a}}(\alpha_j) = \mathrm{e}^{\mathrm{i}\mathbf{a}\cdot\boldsymbol{\sigma}\,\alpha_j/2} = I_{2\times2}\cos\frac{\alpha_j}{2} + \mathrm{i}\,(\mathbf{a}\cdot\boldsymbol{\sigma})\sin\frac{\alpha_j}{2}, \tag{3.15}$$

where $\mathbf{a}\cdot\boldsymbol{\sigma} = a_x\sigma_x + a_y\sigma_y + a_z\sigma_z$ involves the standard Pauli matrices.

Circuit Interpretation

Intuitively, $F_m^k(\mathbf{a},\boldsymbol{\alpha})$ applies 2^k conditional rotations on the target qubit, one for each configuration of the k controls. For example, when $k=3$, there are eight such controlled operations.

Figure 8 illustrates: - **(a)** the abstract definition of the uniformly controlled rotation, and - **(b)** its explicit gate decomposition using CNOTs and single-qubit rotations.

The effective rotation angles θ_j in the decomposed circuit are related to the original angles α_j via a linear transformation:

$$\begin{bmatrix} \theta_1 \\ \vdots \\ \theta_{2^k} \end{bmatrix} = \mathbf{M} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_{2^k} \end{bmatrix}, \qquad M_{ij} = 2^{-k}(-1)^{b_{j-1} \cdot g_{i-1}}, \tag{3.16}$$

where b_m and g_m denote the **binary code** and the **binary reflected Gray code** representations of the integer m. As shown in Eq. 3.16, the matrix $\mathbf{M}$ is a normalized Walsh-Hadamard transform composed with Gray-code reordering.

Intuition. The Gray-code transformation reorganizes control configurations so that consecutive rotations differ by only one control bit, minimizing the number of required CNOT gates in the circuit implementation.

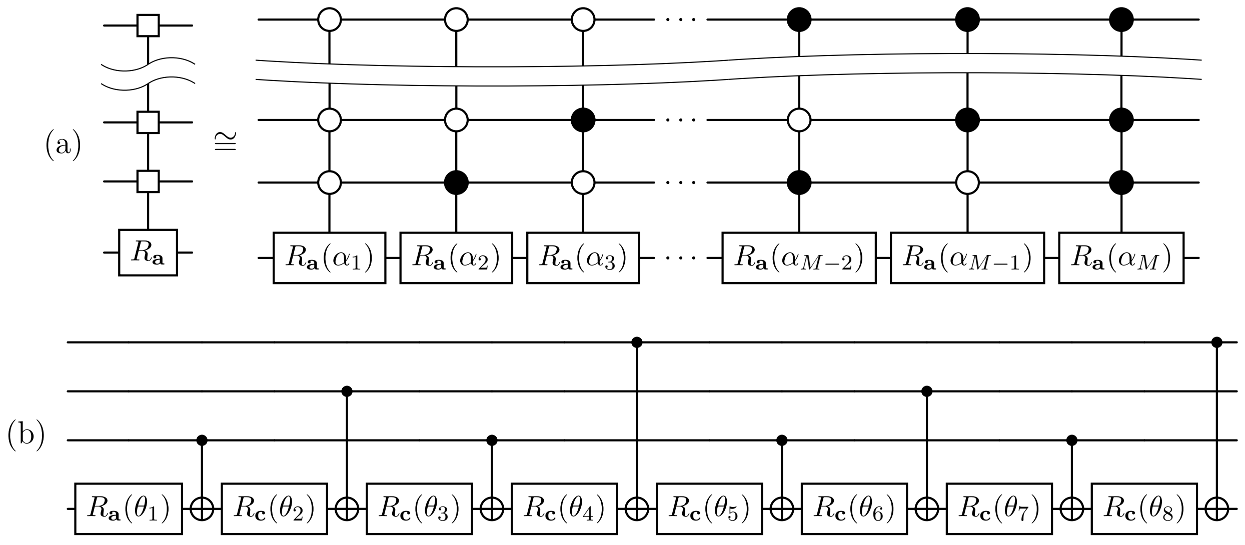

(a)
$R_{\mathbf{a}}$
$\cong$
$R_{\mathbf{a}}(\alpha_1)$
$R_{\mathbf{a}}(\alpha_2)$
$R_{\mathbf{a}}(\alpha_3)$
$R_{\mathbf{a}}(\alpha_{M-2})$
$R_{\mathbf{a}}(\alpha_{M-1})$
$R_{\mathbf{a}}(\alpha_M)$
(b)
$R_{\mathbf{a}}(\theta_1)$
$R_{\mathbf{c}}(\theta_2)$
$R_{\mathbf{c}}(\theta_3)$
$R_{\mathbf{c}}(\theta_4)$
$R_{\mathbf{c}}(\theta_5)$
$R_{\mathbf{c}}(\theta_6)$
$R_{\mathbf{c}}(\theta_7)$
$R_{\mathbf{c}}(\theta_8)$

Figure 8: Quantum circuits for uniformly controlled rotations. (a) Definition of $F_m^k(\mathbf{a}, \boldsymbol{\alpha})$ with k control qubits. (b) Full decomposition for $F_4^3(\mathbf{a}, \boldsymbol{\alpha})$. Angles θ_j are related to the original α_j via Eq. (3.16). Open (filled) circles represent control bits with value 0 (1).

Application: Quantum State Preparation

Goal of this subsection. We construct a unitary that maps the vacuum state directly to an arbitrary target state by separately encoding phases and amplitudes using uniformly controlled rotations.

Uniformly controlled rotations form the building blocks of an efficient algorithm for preparing arbitrary quantum states. Given a normalized n-qubit target state

$$|\psi\rangle = \Big(|c_1|\mathrm{e}^{\mathrm{i}\omega_1}, |c_2|\mathrm{e}^{\mathrm{i}\omega_2}, \dots, |c_N|\mathrm{e}^{\mathrm{i}\omega_N}\Big)^\top, \quad N = 2^n, \tag{3.17}$$

we seek a unitary U such that $U|0\dots0\rangle = |\psi\rangle$.

The process proceeds in two conceptual steps:

(1) Phase Equalization

We first remove all relative phases among the amplitudes by applying a sequence of **uniformly controlled z-rotations**. This transforms $|\psi\rangle$ into a real, positive vector of amplitudes:

$$\mathcal{R}_{\mathbf{z}}|\psi\rangle = \mathrm{e}^{\mathrm{i}\Omega}(|c_1|, |c_2|, \dots, |c_N|)^\top. \tag{3.18}$$

The operator $\mathcal{R}_{\mathbf{z}}$ is built as

$$\mathcal{R}_{\mathbf{z}} = \prod_{k=1}^{n} F_k^{k-1}(\mathbf{z}, \boldsymbol{\alpha}_{n-k+1}^z) \otimes I_{2^{n-k}}, \tag{3.19}$$

with angles

$$(\alpha_j)_k^{\mathbf{z}} = \sum_{l=1}^{2^{k-1}} \frac{\omega_{(2j-1)2^{k-1}+l} - \omega_{(j-1)2^k+l}}{2^{k-1}}, \quad j = 1, \dots, 2^{n-k}. \tag{3.20}$$

Intuition. Phase equalization removes all relative complex phases first, reducing the problem to preparing a real, nonnegative amplitude vector that can be handled solely with y-axis rotations.

(2) Amplitude Encoding

Next, we use **uniformly controlled y-rotations** to redistribute amplitudes. Each rotation step eliminates amplitudes corresponding to basis states with a 1 in a specific qubit position, progressively collapsing the superposition toward $|0 \dots 0\rangle$:

$$\mathcal{R}_{\mathbf{y}} \mathcal{R}_{\mathbf{z}} |\psi\rangle = \mathrm{e}^{\mathrm{i}\Phi} |0 \dots 0\rangle . \tag{3.21}$$

For instance, the first rotation in this sequence is $F_n^{n-1}(\mathbf{y}, \boldsymbol{\alpha}^y)$, where

$$\alpha_j^y = 2 \arcsin \left(\frac{|c_{2j}|}{\sqrt{|c_{2j-1}|^2 + |c_{2j}|^2}} \right), \tag{3.22}$$

which cancels all amplitudes with the least significant qubit set to 1. After this, the last qubit is disentangled and initialized to $|0\rangle$:

$$F_n^{n-1}(\mathbf{y}, \boldsymbol{\alpha}^y) |\psi\rangle = \mathrm{e}^{\mathrm{i}\Omega} \Big(\sqrt{|c_1|^2 + |c_2|^2}, \sqrt{|c_3|^2 + |c_4|^2}, \dots \Big)^{\top} \otimes |0\rangle . \tag{3.23}$$

Intuition. Each uniformly controlled y-rotation redistributes probability mass so that basis states with a 1 in a chosen qubit are eliminated, progressively disentangling qubits from least to most significant.

Repeating this procedure over all n qubits yields:

$$\mathcal{R}_{\mathbf{y}} = \prod_{k=1}^{n} F_k^{k-1}(\mathbf{y}, \boldsymbol{\alpha}_{n-k+1}^y) \otimes I_{2^{n-k}}, \tag{3.24}$$

with

$$(\alpha_j)_k^{\mathbf{y}} = 2 \arcsin \left(\frac{\sqrt{\sum_{l=1}^{2^{k-1}} |c_{(2j-1)2^{k-1}+l}|^2}}{\sqrt{\sum_{l=1}^{2^k} |c_{(j-1)2^k+l}|^2}} \right). \tag{3.25}$$

Recovering the Target State

To reconstruct $|\psi\rangle$ from the vacuum state, apply the inverse transformation:

$$|\psi\rangle = \mathcal{R}_{\mathbf{z}}^{\dagger}\mathcal{R}_{\mathbf{y}}^{\dagger}\mathrm{e}^{\mathrm{i}\Phi}\,|0\ldots0\rangle\,. \tag{3.26}$$

Intuition. Reversing the disentangling sequence reconstructs the target state from the vacuum, yielding a compact, non-recursive circuit with analytically determined parameters.

Checkpoint Summary: Uniformly Controlled Rotations (UCR)

What problem did we solve? We introduced an algebraic state-preparation method based on uniformly controlled rotations that separately encodes phases and amplitudes using analytically defined rotation networks.

When should I use this method? Use this approach when circuit compactness, predictable gate counts, and compiler-friendly structure are priorities.

What is the asymptotic cost?

CNOT gates: $2^{n+2} - 4n - 4$; Single-qubit rotations: $2^{n+2} - 5$

4 Decomposition of an Arbitrary Unitary Gate

Implementing an **arbitrary unitary** U on an n-qubit register entails **decomposing** the $[2^n \times 2^n]$ matrix into a product of native one- and two-qubit gates. The goal is to factor U into building blocks whose circuit realizations are known and hardware-efficient. Since CNOT gates are required to generate entanglement but also increase depth and error susceptibility, most algorithms seek to minimize their count. Several families of methods achieve this via *controlled/multiplexed* gates and *recursive* constructions that mirror the state-preparation techniques of Section 3.

4.1 Decomposition Using Givens Rotations

Goal of this subsection. We describe how an arbitrary n-qubit unitary can be decomposed into a sequence of two-level Givens rotations that eliminate matrix elements column by column, closely mirroring the QR decomposition.

A particularly transparent synthesis strategy is based on **Givens rotations**,[4] which provide a direct quantum analogue of classical QR factorization. The central idea is to eliminate off-diagonal matrix elements one column at a time using two-level unitaries that act nontrivially only within a selected two-dimensional subspace. Let $N = 2^n$. For a fixed column index i, a Givens rotation ${}^{i}G_{j,k}$ acts on the computational basis states $|j\rangle$ and $|k\rangle$ so as to zero the matrix element U_{ji} by mixing it with U_{ki}. The corresponding 2×2 unitary block embedded in the full $[N \times N]$ matrix is

$$
{}^{i}\Gamma_{j,k} \;=\; \frac{1}{\sqrt{|U_{ji}|^2 + |U_{ki}|^2}} \begin{pmatrix} U^*_{ki} & U^*_{ji} \\ -U_{ji} & U_{ki} \end{pmatrix}, \tag{4.1}
$$

which is placed into the (j, k) rows and columns of the identity to form the full Givens operator ${}^{i}G_{j,k}$.

$$
{}^{i}G_{j,k} = \begin{array}{cccccc} & \overset{k}{\downarrow} & & \overset{j}{\downarrow} & & \\ \ddots\, 1 \,\ddots & & & & & \\ & U^*_{ki} & & U^*_{ji} & & \leftarrow k \\ & & \ddots\, 1 \,\ddots & & & \\ & -U_{ji} & & U_{ki} & & \leftarrow j \\ & & & & \ddots\, 1 \,\ddots & \end{array} \tag{4.2}
$$

Intuition. Each Givens rotation acts only on a two-dimensional subspace, using one matrix element to cancel another while leaving all other basis states unchanged.

The decomposition proceeds column by column. One first removes all entries below the top element of the first column by applying ${}^{1}G_{N,N-1}$, then ${}^{1}G_{N-1,N-2}$, and so on down to ${}^{1}G_{2,1}$.

Writing

$$\begin{aligned} U_1 &= {}^{1}G_{N,N-1}\, U, \\ U_2 &= {}^{1}G_{N-1,N-2}\, U_1, \\ &\;\vdots \\ U_{N-1} &= {}^{1}G_{2,1}\, U_{N-2}, \end{aligned} \tag{4.3}$$

the first column becomes $(1, 0, \ldots, 0)^{\top}$ up to a global phase. Repeating the same idea for subsequent columns triangularizes the matrix. To make the result exactly the identity rather than a trivial phase multiple, one first adjusts the global phase by multiplying U with $\mathrm{e}^{-\mathrm{i}\arg(\det U)/N} I$. A concrete N=4 example is shown in Fig. 9.

$$U \xrightarrow{\mathrm{e}^{-\mathrm{i}\arg(\det U)/4} I} \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}$$

$$\xrightarrow{{}^{1}G_{4,3}} \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ 0 & * & * & * \end{pmatrix} \xrightarrow{{}^{1}G_{3,2}} \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix} \xrightarrow{{}^{1}G_{2,1}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix}$$

$$\xrightarrow{{}^{2}G_{4,3}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \end{pmatrix} \xrightarrow{{}^{2}G_{3,2}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix} \xrightarrow{{}^{3}G_{4,3}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$U = \exp\left[\mathrm{i}\arg\left(\det U\right)/4\right] I\, {}^{1}G^{\dagger}_{4,3}\, {}^{1}G^{\dagger}_{3,2}\, {}^{1}G^{\dagger}_{2,1}\, {}^{2}G^{\dagger}_{4,3}\, {}^{2}G^{\dagger}_{3,2}\, {}^{3}G^{\dagger}_{4,3}$$

Figure 9: Schematic representation of the QR decomposition of a [4×4] unitary matrix U. The asterisks denote the arbitrary complex numbers.

In general, the full factorization can be written compactly as

$$\left[\prod_{i=1}^{N-1}\prod_{j=i+1}^{N}{}^{N-i}G_{j,j-1}\right]\left(\mathrm{e}^{-\mathrm{i}\arg(\det U)/N}I\right)U \;=\; I, \tag{4.4}$$

which implies

$$U \;=\; \mathrm{e}^{\mathrm{i}\arg(\det U)/N}\,I\prod_{i=1}^{N-1}\prod_{j=1}^{N-i}{}^{i}G^{\dagger}_{N-j+1,\,N-j}. \tag{4.5}$$

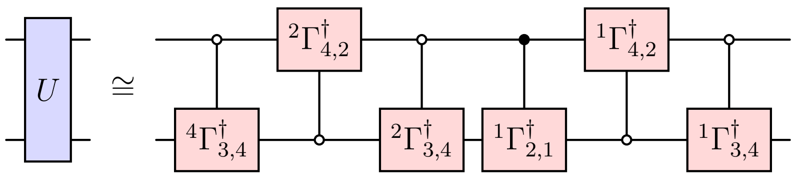
Intuition. Each two-level rotation eliminates a single matrix element while leaving previously fixed entries unchanged. Repeating this process column by column systematically reduces the unitary to the identity. Reversing the sequence of rotations then reconstructs the original operator as an ordered product of simple, localized two-level transformations.

For quantum circuits, the ordering of basis states is crucial. With standard binary ordering, many ${}^{i}G_{j,k}$ couple basis vectors that differ in multiple qubits, making a direct realization in terms of elementary gates cumbersome. **Gray-code** ordering remedies this by arranging basis states so that neighbors differ in exactly one bit. Under Gray ordering, each ${}^{i}G_{j,k}$ acts on adjacent basis states and can be implemented as a single-qubit rotation with $n-1$ control qubits. Denoting by $\gamma(\cdot)$ the Gray-code permutation (i.e., the integer value of a Gray-coded bitstring), the factorization in Eq. (4.4) becomes[4]

$$\left[\prod_{i=1}^{N-1}\prod_{j=i+1}^{N}{}^{\gamma(N-i)}G_{\gamma(j),\,\gamma(j-1)}\right]\left(\mathrm{e}^{-\mathrm{i}\arg(\det U)/N}I\right)U \;=\; I. \tag{4.6}$$

Intuition. Gray-code ordering ensures that each two-level rotation couples basis states differing by a single bit, allowing direct realization as multi-controlled single-qubit gates.

A two-qubit example is depicted in Fig. 10, where the Givens rotations ${}^{i}\Gamma^{\dagger}_{j,k}$ in Gray order translate into fully controlled single-qubit gates. Each such multi-controlled Givens rotation can be decomposed into a ZYZ sequence (R_z, R_y, R_z) and an $O(n)$ overhead of CNOTs to implement the controls. The optimized construction of Ref. 4 achieves a CNOT complexity $O(4^n)$, matching known asymptotic lower bounds for generic n-qubit unitaries.



U
$\cong$
${}^{2}\Gamma^{\dagger}_{4,2}$
${}^{4}\Gamma^{\dagger}_{3,4}$
${}^{2}\Gamma^{\dagger}_{3,4}$
${}^{1}\Gamma^{\dagger}_{2,1}$
${}^{1}\Gamma^{\dagger}_{4,2}$
${}^{1}\Gamma^{\dagger}_{3,4}$

Figure 10: QR-style decomposition of a 2-qubit unitary U using Givens rotations ${}^{i}\Gamma^{\dagger}_{j,k}$ ((4.1)) in Gray-code ordering. Adjacent basis states differ by one bit, enabling realization as multi-controlled single-qubit rotations.

Code overview. The accompanying Python snippets (Script S.2.1–Script S.2.3) implement the construction of two-level Givens rotations and their sequential application to triangularize U. The optional Gray-coded mode makes the rotation sequence directly amenable to circuit synthesis with multi-controlled single-qubit gates.

This formulation makes explicit how an arbitrary unitary breaks down into a product of two-level operations that, under Gray ordering, become multi-controlled single-qubit rotations with known ZYZ decompositions and well-characterized CNOT overhead, thereby providing a clear pathway from matrix factorization to hardware-level circuits on NISQ devices.

Example: Decomposition of a Random Unitary Matrix To verify the implementation, Script S.2.4 defines a Haar-random 4×4 unitary U using `scipy.stats.unitary_group` and applies the routine `Gmatrix()` defined above. By construction, `Gmatrix` returns $R\,U_{\text{phase}}$ with a global phase adjustment $U_{\text{phase}} = \mathrm{e}^{-\mathrm{i}\arg(\det U)/N} I$ (here N=4), so that

$$(R\,U_{\text{phase}})\,U \;=\; R\,(U_{\text{phase}}U) \;=\; I \tag{4.7}$$

up to machine precision. The printout below shows the ordered Givens sequence and confirms that the transformed matrix is (numerically) the identity.

Discussion. The printed sequence lists the two-level Givens unitaries in the order they are applied to eliminate subdiagonal entries. The scalar D is the global phase adjustment ensuring $\det(DU) = 1$. The same procedure extends directly to larger $N = 2^n$, with Gray ordering making each two-level unitary act on adjacent basis states that differ by one bit, which maps cleanly to multi-controlled single-qubit rotations.

4.1.1 Real-parameter Givens variant

Goal of this subsection. We show how complex Givens rotations can be replaced by purely real rotations by separating local phase corrections, thereby simplifying circuit synthesis.

The two-level complex blocks ${}^{i}\Gamma_{j,k}$ defined in Eq. (4.1) can be recast as real rotations by first applying a *local phase correction* to the two participating basis states. Specifically, we introduce a diagonal unitary ${}^{i}\tilde{\Delta}_{j,k}$ that multiplies rows j and k by $\mathrm{e}^{-\mathrm{i}\phi_{ji}}$ and $\mathrm{e}^{-\mathrm{i}\phi_{ki}}$, where $\phi_{ji} = \arg(U_{ji})$ and $\phi_{ki} = \arg(U_{ki})$. After this transformation, the relevant column entries become real (and nonnegative, if desired), allowing the elimination step to be performed using a real 2×2 rotation,

$$
{}^{i}\tilde{\Gamma}_{j,k} = \begin{pmatrix} \cos\theta_{ijk} & \sin\theta_{ijk} \\ -\sin\theta_{ijk} & \cos\theta_{ijk} \end{pmatrix}, \qquad \cos\theta_{ijk} = \frac{\mathrm{Re}\; U'_{ki}}{\sqrt{(\mathrm{Re}\; U'_{ji})^2 + (\mathrm{Re}\; U'_{ki})^2}}, \tag{4.8}
$$

where $U' = {}^{i}\tilde{\Delta}_{j,k}\, U$ denotes the phase-corrected matrix, ensuring that the pair (U'_{ji}, U'_{ki}) is real.

The full $[N \times N]$ embedding ${}^{i}\tilde{G}_{j,k}$ of the real two-level block ${}^{i}\tilde{\Gamma}_{j,k}$ is obtained by inserting it into the (j, k) rows and columns of the identity, analogously to Eq. (4.2). Explicitly,

$$
{}^{i}\tilde{G}_{j,k} = \begin{array}{cc}
\begin{array}{ccccc} & \overset{k}{\downarrow} & & \overset{j}{\downarrow} & \end{array} & \\
\left(\begin{array}{ccccc}
\ddots 1 \ddots & & & & \\
 & \cos\theta_{ijk} & & \sin\theta_{ijk} & \\
 & & \ddots 1 \ddots & & \\
 & -\sin\theta_{ijk} & & \cos\theta_{ijk} & \\
 & & & & \ddots 1 \ddots
\end{array}\right) &
\begin{array}{c} \\ \leftarrow k \\ \\ \leftarrow j \\ \\ \end{array}
\end{array} . \tag{4.9}
$$

Practical note. Beyond reducing circuit complexity, the real-parameter Givens variant substantially simplifies *parameter synthesis* by eliminating complex-valued angles in favor of purely real rotation parameters. This formulation is therefore preferable when targeting hardware with native real-valued rotations, restricted control precision, or compilation pipelines that benefit from reduced parameter overhead.

The overall factorization then interleaves these phase corrections and real rotations, with the global det-phase written for general dimension N (not fixed to 4):

$$
{}^{N}\tilde{\Delta}_{N,N} \left[\prod_{i=1}^{N-1} \prod_{j=i+1}^{N} {}^{N-i}\tilde{G}_{j,j-1} \; {}^{N-i}\tilde{\Delta}_{j,j-1} \right] \left(\mathrm{e}^{-\mathrm{i}\arg(\det U)/N} I \right) U \;=\; I, \tag{4.10}
$$

hence

$$
U \;=\; \mathrm{e}^{\mathrm{i}\arg(\det U)/N} I \left[\prod_{i=1}^{N-1} \prod_{j=1}^{N-i} {}^{i}\tilde{\Delta}^{\dagger}_{N-j+1,\,N-j} \; {}^{i}\tilde{G}^{\dagger}_{N-j+1,\,N-j} \right] {}^{N}\tilde{\Delta}^{\dagger}_{N,N}. \tag{4.11}
$$

As in the complex-valued construction, Gray-code ordering may be applied to ensure that each two-level block acts on adjacent basis states, enabling a direct mapping to multi-controlled single-qubit rotations with standard ZYZ decompositions.

Checkpoint Summary: Givens (QR-Style) Unitary Decomposition

What problem did we solve? We decomposed an arbitrary n-qubit unitary into a sequence of two-level Givens rotations that eliminate matrix elements column by column, mirroring QR decomposition.

When should I use this method? Use this as a transparent baseline/pedagogical reference or for validating more optimized synthesis strategies.

What is the asymptotic cost? CNOT gates: $O(4^n)$
Single-qubit rotations: $O(4^n)$

4.2 Column-by-Column Decomposition

The idea is to apply the **reverse state-preparation** technique successively to each column of the target unitary until the whole matrix becomes the identity. Let $U_0 := U$. At the first stage, one constructs a state-preparation unitary $\mathcal{R}_0$ that maps the first column of U_0 to the computational basis state $|0\rangle$; equivalently,

$$\mathcal{R}_0 \, U_0 \, |0\rangle = |0\rangle \, , \tag{4.12}$$

so that in the updated matrix $U_1 := \mathcal{R}_0 U_0$ the first column is $(1, 0, \ldots, 0)^\top$. The second stage builds $\mathcal{R}_1$ such that

$$\mathcal{R}_1 \, U_1 \, |1\rangle = |1\rangle \, , \tag{4.13}$$

while preserving the previous column, i.e. $\mathcal{R}_1 |0\rangle = |0\rangle$. Proceeding in this way, the $(j+1)$-st stage constructs $\mathcal{R}_j$ so that

$$\mathcal{R}_j \, U_j \, |j\rangle = |j\rangle \, , \qquad U_{j+1} := \mathcal{R}_j U_j, \tag{4.14}$$

and simultaneously enforces $\mathcal{R}_j \left|i\right\rangle = \left|i\right\rangle$ for all $i < j$. After $2^n{-}1$ stages the updated matrix U_{2^n-1} is diagonal, and the entire decomposition reads

$$U \;=\; \mathcal{R}_0^\dagger \, \mathcal{R}_1^\dagger \, \cdots \, \mathcal{R}_{2^n-2}^\dagger \, D^\dagger, \tag{4.15}$$

with the final operator $D = \mathcal{R}_{2^n-1}^\dagger$ diagonal.

Constructing the column maps. Each $\mathcal{R}_j$ may be realized either with **quantum multiplexors** (cf. Section 3.1) or with **uniformly controlled rotations** (Section 3.2). Naively applying these state-preparation blocks does not, however, guarantee the invariance constraints $\mathcal{R}_j \left|i\right\rangle = \left|i\right\rangle$ for $i < j$. The **modified column-by-column decomposition (CCD)** of Iten *et al.*[6] enforces these constraints with minimal overhead: after placing the required uniformly controlled gates (UCGs) or multiplexors to align the $(j{+}1)$-st column, one inserts a small number of **multi-controlled gates** (MCGs) that cancel any unintended action on the previously fixed columns. In practice, this correction stage uses the fact that the already prepared columns define projectors onto $\left|0\right\rangle, \ldots, \left|j-1\right\rangle$; controls on those subspaces protect what has been fixed while steering only the current column.

Three-qubit illustration. For a three-qubit register, Fig. 11 shows the progressive elimination in the first column (top) and the corresponding circuit (bottom). The red blocks are sequential multiplexed rotations $R^{(i)} \equiv R_y^{(i)} R_z^{(i)}$ (this order is used throughout) that perform the reverse state preparation; the green block D is diagonal and collects phases. In Fig. 12, the second column is prepared while preserving the first: the blue blocks are the additional MCGs that correct the collateral action of the red UCGs on $\left|0\right\rangle$.

$$U|0\rangle = \left[\begin{array}{c} * \\ * \\ \hline * \\ * \\ \hline * \\ * \\ \hline * \\ * \end{array}\right] \longrightarrow \left[\begin{array}{c} * \\ 0 \\ \hline * \\ 0 \\ \hline * \\ 0 \\ \hline * \\ 0 \end{array}\right] \longrightarrow \left[\begin{array}{c} * \\ 0 \\ 0 \\ 0 \\ \hline * \\ 0 \\ 0 \\ 0 \end{array}\right] \longrightarrow \left[\begin{array}{c} * \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}\right] \longrightarrow \left[\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}\right]$$

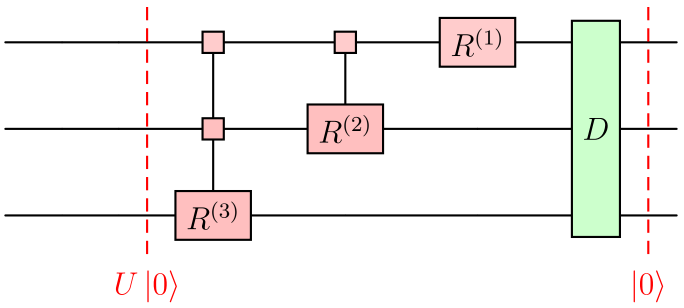


Figure 11: Step 0: transforming the first column to $|0\rangle$. Top: successive nulling of sub-entries; asterisks denote arbitrary complex numbers. Bottom: reverse state-preparation with multiplexed rotations $R^{(i)} \equiv R_y^{(i)} R_z^{(i)}$ (red) and a diagonal phase block D (green), cf. (3.7).

General stage j (harmonized notation). Throughout, products are written as a left-to-right ordered product using $\overrightarrow{\prod}$. For an n-qubit register,

$$\mathcal{R}_j = \overrightarrow{\prod_{s=0}^{n-1}} \left[\Delta^{(n-s)} \otimes I^{\otimes s}\right] \left[R_j^{(n-s)} \otimes I^{\otimes s}\right] \left[\tilde{R}_j^{(n-s)}\right], \tag{4.16}$$

where $R_j^{(n-s)}$ is a UCG (multiplexor) on target qubit $(n-s)$ with universal controls on the s less-significant qubits and uses the rotation order $R_y R_z$; $\tilde{R}_j^{(n-s)}$ is an MCG on the same target with controls on the remaining $n-1$ qubits; and $\Delta^{(n-s)}$ is a diagonal gate acting on the $(n-s)$ least-significant qubits. The associated schematic circuit is shown in Fig. 13. When $s = n-1$, the control-free $R^{(1)}$ is redundant and can be replaced by the identity.

Complexity. The full column-by-column decomposition of a generic n-qubit unitary requires $4^n + O(n^2)2^n$ CNOTs. This already improves on the straightforward QR-style ap-

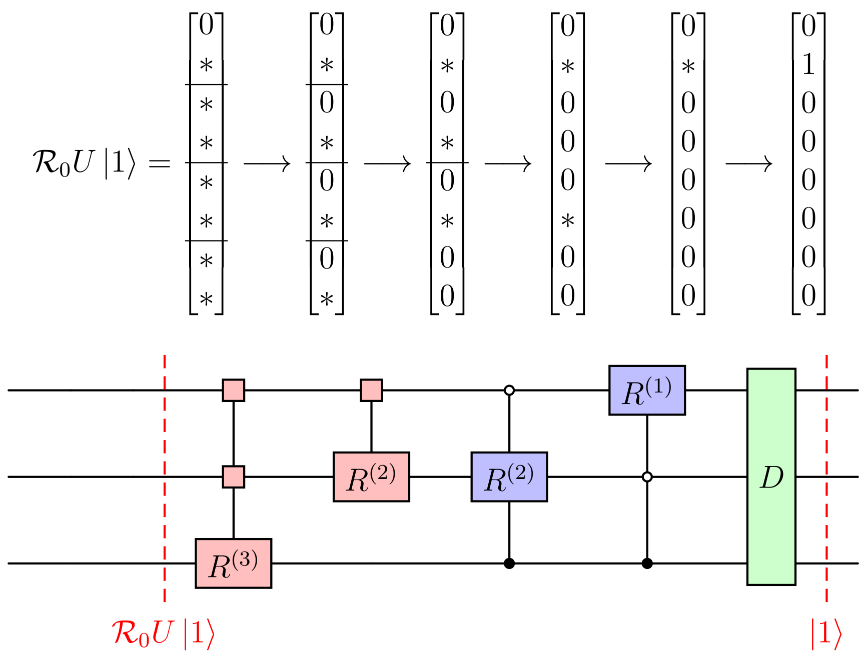


Figure 12: Step 1: preparing the second column while preserving the first. Red: UCG/multiplexed rotations $R^{(i)}$. Blue: correcting MCGs that protect $|0\rangle$. Green: diagonal D.

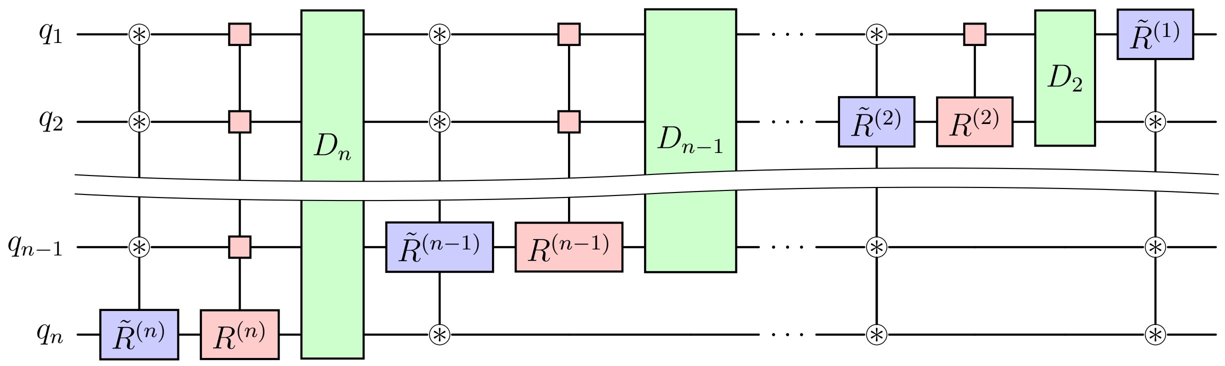


Figure 13: One stage of the column-by-column algorithm in the form of Eq. (4.16). Color code: UCG/multiplexors (red, using $R_y R_z$ order), correcting MCGs (blue), and diagonal gates (green). Circled asterisks indicate zero or one control bits depending on context.

proach, and further optimizations such as the **Quantum Shannon Decomposition** (QSD) and recursive **Cosine-Sine Decomposition** (CSD) reduce constants and refine structure. A summary comparison of CNOT counts for QR, CCD, and QSD is reported in Table 2.

Intuition Summary. Conceptually, CCD performs reverse state preparation on each column while protecting previously fixed columns via additional controls.

4.3 Recursive Cosine–Sine and Quantum Shannon Decompositions

The synthesis methods introduced so far reduce arbitrary unitaries to sequences of controlled operations, but they do not fully exploit the hierarchical block structure inherent in large multi-qubit operators. In this section, we introduce two closely related recursive factorizations-the **Cosine–Sine Decomposition (CSD)**[5] and the **Quantum Shannon Decomposition (QSD)**[2]-that systematically partition an n-qubit unitary into smaller, structured blocks that map naturally to quantum circuits. Through recursive application, these decompositions confine all control complexity to multiplexed single-qubit rotations while relegating unconstrained unitary blocks to lower-dimensional subspaces. The resulting circuit constructions achieve significantly reduced CNOT counts and depth compared to more direct synthesis strategies, making CSD and QSD the methods of choice for near-optimal implementation of large unitaries on both NISQ and fault-tolerant quantum architectures.

Cosine-Sine Decomposition (CSD). For an even-dimensional $[2^n \times 2^n]$ unitary U (an n-qubit gate), the CSD expresses U as

$$U = \left(\begin{array}{c|c} U_{00} & U_{01} \\ \hline U_{10} & U_{11} \end{array}\right) = \underbrace{\left(\begin{array}{c|c} A_0 & 0 \\ \hline 0 & A_1 \end{array}\right)}_{A_0 \oplus A_1} \underbrace{\left(\begin{array}{c|c} C & -S \\ \hline S & C \end{array}\right)}_{\text{cos-sin block}} \underbrace{\left(\begin{array}{c|c} B_0 & 0 \\ \hline 0 & B_1 \end{array}\right)}_{B_0 \oplus B_1}, \tag{4.17}$$

where $A_0, A_1, B_0, B_1 \in \mathrm{U}(2^{n-1})$. The diagonal matrices $C = \mathrm{diag}(\cos\frac{\theta_1}{2}, \ldots, \cos\frac{\theta_{2^{n-1}}}{2})$ and $S = \mathrm{diag}(\sin\frac{\theta_1}{2}, \ldots, \sin\frac{\theta_{2^{n-1}}}{2})$ satisfy $C^2 + S^2 = I_{2^{n-1}}$, so the middle factor is unitary. In circuit terms, $A_0 \oplus A_1$ and $B_0 \oplus B_1$ are $(n-1)$-qubit *multiplexors* controlled by the most significant qubit, and the cos-sin block implements a *multiplexed* single-qubit rotation $R_y(\boldsymbol{\theta})$ on the most significant qubit, conditioned on the remaining $n-1$ qubits:

$$C - \mathrm{i}S \;\equiv\; \exp\Big(-\frac{\mathrm{i}}{2}\,\theta(\mathbf{c})\,\sigma_y\Big) \quad \text{with} \quad \mathbf{c} \in \{0,1\}^{n-1}. \tag{4.18}$$

A schematic circuit is shown in Fig. 14.

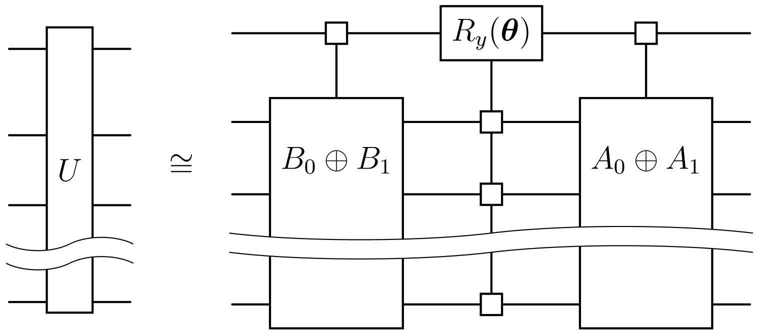


Figure 14: One CSD layer: a multiplexed $R_y(\boldsymbol{\theta})$ on the most significant qubit sandwiched by $(n-1)$-qubit multiplexors $A_0 \oplus A_1$ and $B_0 \oplus B_1$.

Recursive CSD. Applying CSD recursively to $A_0 \oplus A_1$ and $B_0 \oplus B_1$ yields a hierarchy down to single-qubit multiplexors, as sketched in Fig. 15. Final demultiplexing into elementary gates can be done in several analytically tractable ways; representative optimized realizations achieve CNOT counts on the order of 4^n with improved constants compared to QR-based schemes (see Table 2 for a numerical comparison).

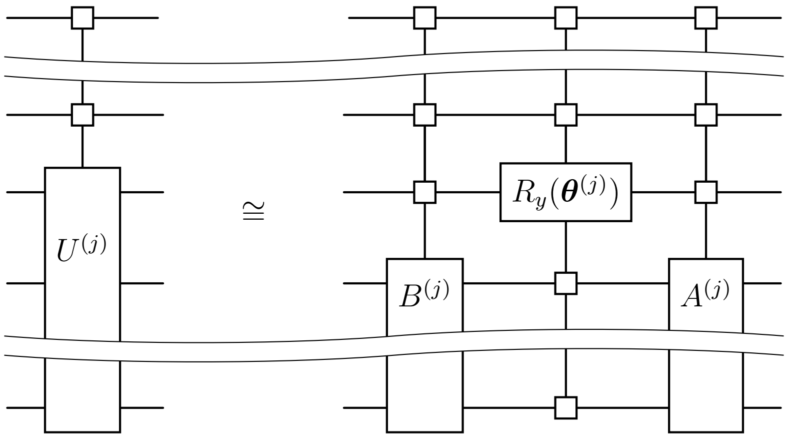


Figure 15: Recursive CSD step: each multiplexor is further decomposed into a smaller multiplexed rotation flanked by smaller multiplexors, until only single-qubit blocks remain.

Quantum Shannon Decomposition (QSD). QSD sharpens the multiplexor decomposition further. Suppose

$$U = \left(\begin{array}{c|c} U_0 & 0 \\ \hline 0 & U_1 \end{array}\right) \tag{4.19}$$

is a multiplexor controlled by the most significant qubit. Diagonalize the $(n-1)$-qubit unitary $U_0 U_1^\dagger$ as

$$U_0 U_1^\dagger = V\, D^2\, V^\dagger, \tag{4.20}$$

with V unitary and D diagonal unitary (so D^2 carries the eigenvalues). Then

$$U = \left(\begin{array}{c|c} V & 0 \\ \hline 0 & V \end{array}\right)\left(\begin{array}{c|c} D & 0 \\ \hline 0 & D^\dagger \end{array}\right)\left(\begin{array}{c|c} W & 0 \\ \hline 0 & W \end{array}\right), \qquad W = D\, V^\dagger U_1. \tag{4.21}$$

The middle block $D \oplus D^\dagger$ is a *diagonal* multiplexor and maps to a single multiplexed $R_z(\boldsymbol{\phi})$ on the most significant qubit, while $V \oplus V$ and $W \oplus W$ are $(n-1)$-qubit unitaries applied independently of the control. The circuit form is shown in Fig. 16.

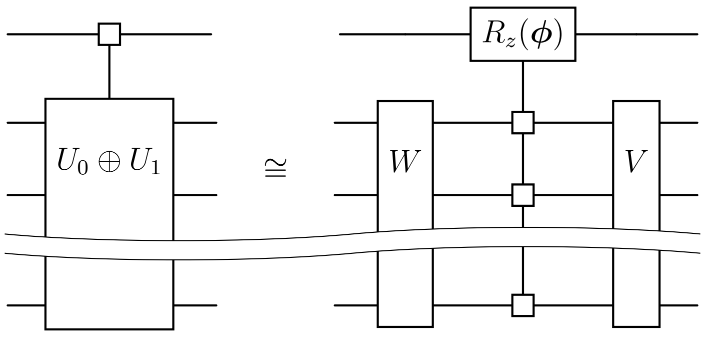
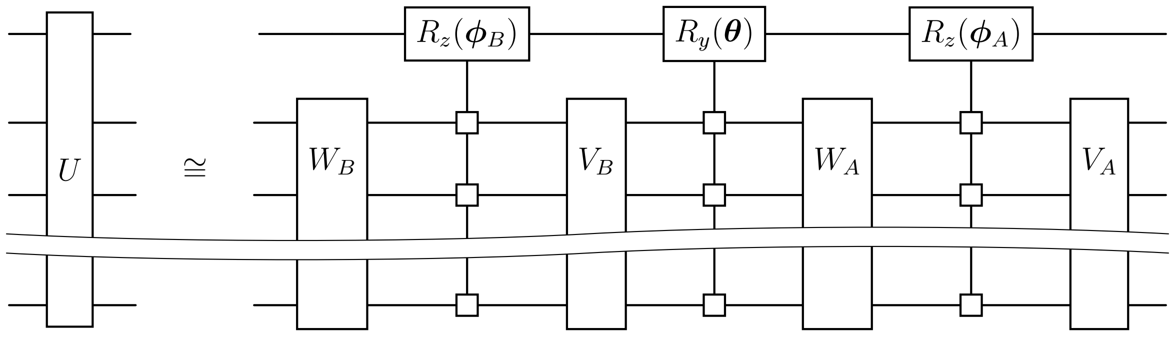


Figure 16: QSD of a single-control multiplexor: two identical $(n-1)$-qubit unitaries V, W flanking a multiplexed $R_z(\boldsymbol{\phi})$ on the most significant qubit.

CSD + QSD synthesis. Substituting the QSD factorization for each multiplexor that appears in a CSD layer (Fig. 14) yields a circuit where the controlled structure is confined to *single-qubit* multiplexed rotations $R_y(\boldsymbol{\theta})$ and $R_z(\boldsymbol{\phi})$ on the most significant qubit, while the remaining $(n-1)$-qubit blocks are plain unitaries. An illustrative first recursion step is shown in Fig. 17; subsequent steps continue along the same pattern until only single-qubit gates and diagonal phases remain.



U
$\cong$
$R_z(\boldsymbol{\phi}_B)$
$R_y(\boldsymbol{\theta})$
$R_z(\boldsymbol{\phi}_A)$
W_B
V_B
W_A
V_A

Figure 17: First QSD substitution inside a CSD layer: controlled structure collapses to multiplexed R_z and R_y on the most significant qubit; all other blocks act on the $(n-1)$-qubit subspace.

Asymptotic gate counts. With careful demultiplexing and diagonal synthesis, recursive CSD attains CNOT counts strictly below the 4^n of QR-style schemes, and QSD improves constants further. An optimized QSD pipeline realizes a CNOT count

$$\frac{23}{48}\,4^n \;-\; \frac{3}{2}\,2^n \;+\; \frac{4}{3}, \tag{4.22}$$

while comparable CSD constructions typically scale on the order of 4^n with smaller prefactors than QR. Table 2 summarizes representative counts reported in the literature across n, highlighting the progressive savings from QR to CSD to QSD.

Table 2: Representative elementary-gate counts for QR, CSD, and QSD decompositions of a generic n-qubit unitary.

n	1	2	3	4	5	6	7	8	9
					CNOT gate count				
QR	0	4	64	536	4,156	22,618	108,760	486,052	2,078,668
CSD	0	4	26	118	494	2,014	8,126	32,638	130,814
QSD	0	3	21	105	465	1,953	8,001	32,385	130,305
					Total gate count				
QR	1	14	136	980	7,384	42,390	208,820	944,280	4,062,520
CSD	1	11	58	249	1,016	4,087	16,374	65,525	262,132
QSD	1	10	54	262	1,142	4,758	19,414	78,422	315,222

[a]Data reproduced from Ref. 29.

Checkpoint Summary: Recursive CSD and Quantum Shannon Decomposition (QSD)

What problem did we solve? We reduced arbitrary n-qubit unitaries into recursively structured blocks via CSD and optimized them via Quantum Shannon Decomposition (QSD), confining controlled structure to multiplexed single-qubit rotations.

When should I use this method? Use this for near-optimal synthesis of large unitaries when CNOT count and depth matter (especially on NISQ devices).

What is the asymptotic cost? CNOT gates: CSD: $O(4^n)$ (smaller constants) Optimized QSD: $\frac{23}{48}4^n - \frac{3}{2}2^n + O(1)$

5 Walsh Decomposition of Diagonal Unitaries

General-purpose decompositions often produce deep circuits even for diagonal unitaries, which is undesirable on NISQ devices. Diagonal gates admit a far more efficient synthesis via the Walsh basis.[9–11] Any $[2^n \times 2^n]$ diagonal unitary can be written as

$$U = e^{iF}, \qquad F = \mathrm{diag}(f_0, \ldots, f_{2^n-1}) \ \text{ real.} \tag{5.1}$$

Expanding F in the Pauli basis and noting that only tensor products of $\{I, Z\}$ contribute for diagonal operators gives

$$F = \sum_{j=0}^{2^n-1} a_j\, w_j, \qquad w_j := Z_1^{j_1} \otimes Z_2^{j_2} \otimes \cdots \otimes Z_n^{j_n}, \tag{5.2}$$

where $j = (j_1 \dots j_n)_2$ is the *binary* label, with $j = \sum_{l=1}^{n} j_l 2^{n-l}$ (Most Significant Bit first, MSB-first). The Walsh coefficients follow from Hilbert-Schmidt inner products:

$$a_j = 2^{-n}\,\mathrm{Tr}(w_j F) = 2^{-n} \sum_{k=0}^{2^n-1} (-1)^{\sum_{l=1}^{n} j_l k_l} f_k, \tag{5.3}$$

which is precisely the (dyadic) Walsh-Fourier transform.[7,8] Since all w_j commute, the unitary factors:

$$U = \prod_{j=0}^{2^n-1} \mathrm{e}^{\mathrm{i} a_j w_j}. \tag{5.4}$$

Each factor $\mathrm{e}^{\mathrm{i} a_j w_j}$ is efficiently realized with one single-qubit Z-rotation and at most $2n$ CNOTs:[9,12] for a single Z on a target qubit, $\mathrm{e}^{\mathrm{i}aZ} = R_z(-2a)$.

Commutation allows strong cancellations. Ordering terms by a **Gray code** ensures successive indices differ by one bit, so the entangling "ladders" that compute parity onto a target qubit change minimally between factors, cutting two-qubit cost from naive $O(n2^n)$ to roughly $O(2^n)$ in practice.[9] Below we give a compact, modular implementation.

1. Bit utilities (MSB-first) and Gray code. Script S.3.1 provides helper functions to adopt MSB-first bit vectors to match the analytic conventions above.

2. Fast Walsh-Hadamard transform (FWHT). Script S.3.2 computes $a = \mathrm{WHT}(f)/2^n$ in $O(n2^n)$, replacing the $O(4^n)$ double sum in Eq. (5.3).

3. Gate list generation (Walsh strings). For each nonzero a_j, Script S.3.3 synthesizes $\mathrm{e}^{\mathrm{i} a_j w_j}$ as a parity ladder onto the target (highest 1-bit) followed by $R_z(-2a_j)$ and then uncomputes. Gray-ordering the *sequence of* j's reduces CNOTs between consecutive factors.

4. Circuit build & simple cancellation. Adjacent identical CNOTs cancel; this quick pass removes them, as shown in Script S.3.4. Vendor transpilers will do more.

5. End-to-end test. Script S.3.5 starts from a random diagonal unitary $U = \mathrm{diag}(e^{if_k})$, computes a_j via FWHT, generates and optimizes the Gray-ordered circuit, and prints its depth.

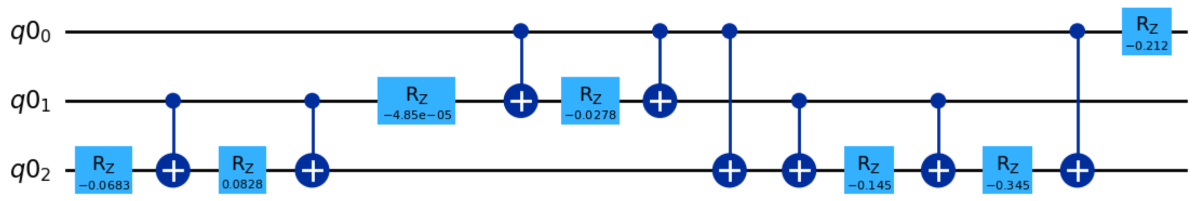


Figure 18: Walsh-based synthesis of a diagonal n=3 unitary. Gray ordering yields a symmetric CNOT pattern with many local cancellations, concentrating cost into a small number of entangling layers.

Discussion. The FWHT computes $\{a_j\}$ in $O(n2^n)$, and the Gray-ordered ladder implementation realizes each factor $e^{ia_jw_j}$ with a single R_z and a short, highly cancellable CNOT stencil. In aggregate this brings the two-qubit cost for a generic diagonal close to $O(2^n)$, substantially shallower than QR/CSD-style compilations. The MSB-first convention in both the analysis and the code ensures that the binary dot product in Eq. (5.3) matches the synthesized control pattern exactly, avoiding subtle off-by-one or endianness bugs.

Practical tip. If you start from a diagonal unitary U instead of phases f_k, call `phases_from_diagonal(np.diag(U))` to get f, then proceed with `walsh_coefficients_from_phases(f)`. Finally, verify numerically that $\prod_j e^{ia_jw_j}$ reproduces U up to machine precision by simulating on the computational basis.

5.1 Exercise: Walsh Synthesis for a Double-Well Potential

This exercise highlights the advantage of Walsh-based synthesis over generic unitary-decomposition pipelines when implementing diagonal unitaries. A detailed solution is provided in Section S.3.6.

We consider a circuit that implements the exact potential-energy phase accumulation

$$|x_j\rangle \longrightarrow \mathrm{e}^{-\mathrm{i}V(x_j)t}\,|x_j\rangle\,, \tag{5.5}$$

which is the central operation in split-operator quantum dynamics for molecular systems.

As discussed in Part II (Sec. 5), Q-SOFT simulations of tunneling, vibrational level splitting, and wavepacket transfer between the two wells alternate applications of the potential-energy propagator $U_V(t) = \mathrm{e}^{-\mathrm{i}tV}$ with the kinetic-energy propagator $U_K(t) = \mathrm{e}^{-\mathrm{i}tK}$. The latter is implemented analogously, since $K = p^2/(2m)$ is diagonal in the momentum basis.

1. Show that the total number of CNOT gates required by the Walsh decomposition is

$$N_{\mathrm{CNOT}} = 2N_2 + 4N_3 + 6N_4, \tag{5.6}$$

   where N_1 is the number of single-qubit Z_i terms, N_2 the number of two-qubit Z_iZ_j terms, N_3 the number of three-qubit $Z_iZ_jZ_k$ terms, and N_4 the number of four-qubit $Z_1Z_2Z_3Z_4$ terms.

2. Compare this result with the number of CNOT gates required to implement (i) a generic four-qubit diagonal unitary and (ii) a fully generic four-qubit unitary, as summarized in Table 2.

Discussion. This exercise demonstrates that for chemically structured Hamiltonians—here, a discretized double-well potential energy surface—Walsh-based (Pauli-string) synthesis exploits diagonal structure to achieve dramatic reductions in entangling-gate counts relative to generic decomposition methods. This advantage grows exponentially with grid size, underscoring the importance of structure-aware synthesis strategies for quantum simulations of molecular dynamics on NISQ and early fault-tolerant devices.

6 Conclusions

This tutorial presented a unified framework for the synthesis of quantum circuits that perform *state initialization* and *unitary decomposition* on n-qubit quantum registers. These two operations-mapping classical data to quantum states and factorizing arbitrary unitaries into executable gate sequences-represent the foundation of nearly all quantum algorithms for chemistry, physics, and information processing. By connecting formal linear-algebraic constructions to explicit circuit realizations, the methods discussed here provide both theoretical insight and practical tools for scalable circuit design.

We began by introducing two efficient approaches to quantum state preparation: the *recursive multiplexor method* and the *uniformly controlled rotation* (UCR) scheme. Both techniques translate amplitude and phase information into sequences of controlled single-qubit rotations, achieving predictable and near-optimal scaling in CNOT gate count. These constructions demonstrate how any target state can be systematically assembled from the vacuum state, forming the building block for more complex quantum transformations.

The second part of the paper addresses the decomposition of general unitary matrices. We examined several complementary methods, including the *Givens rotation (QR) decomposition*, the *column-by-column synthesis*, and the recursive *Cosine-Sine (CSD)* and *Quantum Shannon (QSD)* decompositions. Each algorithm progressively factorizes a large unitary into a product of simpler block-structured operators or multiplexed rotations. The recursive CSD and QSD approaches were shown to yield the most efficient scaling, reducing CNOT counts to $(23/48)\,4^n-(3/2)\,2^n+4/3$-a substantial improvement over the $O(4^n)$ complexity of earlier QR-based methods. The explicit circuits and Python implementations provided in the text demonstrate how these decompositions can be realized and benchmarked in practice.

Finally, we introduced the *Walsh decomposition* as a specialized yet powerful strategy for compiling diagonal unitaries. By expressing diagonal Hamiltonians in the Walsh-Fourier basis and ordering terms according to the Gray code, this method exploits commutation relations among Pauli-Z strings to minimize entangling operations. The resulting circuits require only $O(2^n)$ CNOT gates, making them particularly well suited for noisy intermediate-scale quantum (NISQ) devices where gate depth and coherence time are limiting factors.

Taken together, these methods form a coherent toolkit for quantum circuit synthesis, bridging the gap between mathematical formalism and hardware-level implementation. The modular Python code examples provided through the **QFlux**[1] framework demonstrate how each algorithm can be executed, visualized, and optimized within a unified computational environment. Beyond their immediate application to state preparation and unitary compilation, these techniques lay the groundwork for more advanced protocols-such as time-evolution simulation, variational optimization, and hybrid quantum-classical workflows-covered in subsequent parts of the QFlux series.

In summary, this tutorial establishes a practical foundation for transforming abstract unitaries and quantum states into executable quantum circuits. By combining analytic decompositions with efficient gate synthesis and optimization, it provides researchers and students alike with the tools to design, understand, and implement scalable quantum algorithms on current and future quantum hardware.

Supporting Information

Detailed Jupyter notebooks implementing Q-SOFT and VQTE algorithms, code for Hamiltonian decomposition, and benchmark data are available in the Supporting Information and corresponding Google Colab notebook as well as through the QFlux Documentation site.

Acknowledgements

This work was supported by the National Science Foundation under Award No. 2124511 (CCI Phase I: NSF Center for Quantum Dynamics on Modular Quantum Devices, CQD-MQD) and Award No. 2302908 (Engines Development Award: Advancing Quantum Technologies, CT). The authors also acknowledge the use of IBM Quantum services and open-source software

packages, including Qiskit, Bosonic Qiskit, Strawberry Fields, QuTiP, and MPSQD.

References

(1) Allen, B. C.; Batista, V. S.; Cabral, D. G. A.; Cianci, C.; Dan, X.; Dutta, R.; Geva, E.; Hu, Z.; Kais, S.; Khazaei, P.; Lyu, N.; Mulvihill, E.; Shivpuje, S.; Soudackov, A. V.; Vu, N. P.; Wang, Y.; Wilson, C. QFlux — An Open-Source Python Package for Quantum Dynamics Simulations. https://qflux.batistalab.com, 2025; (accessed: 2025-10-12).

(2) Shende, V. V.; Bullock, S. S.; Markov, I. L. Synthesis of quantum-logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2006**, *25*, 1000–1010.

(3) Mottonen, M.; Vartiainen, J. J.; Bergholm, V.; Salomaa, M. M. Transformation of quantum states using uniformly controlled rotations. 2004; https://arxiv.org/abs/quant-ph/0407010.

(4) Vartiainen, J. J.; Möttönen, M.; Salomaa, M. M. Efficient Decomposition of Quantum Gates. *Physical Review Letters* **2004**, *92*, 177902.

(5) Paige, C. C.; Wei, M. History and generality of the CS decomposition. *Linear Algebra and its Applications* **1994**, *208*, 303–326.

(6) Iten, R.; Colbeck, R.; Kukuljan, I.; Home, J.; Christandl, M. Quantum circuits for isometries. *Physical Review A* **2016**, *93*, 032318.

(7) Walsh, J. L. A Closed Set of Normal Orthogonal Functions. *American Journal of Mathematics* **1923**, *45*, 5–24.

(8) Schipp, F.; Wade, W. R.; Simon, P. *Walsh series. An introduction to dyadic harmonic analysis*; Adam Hilger, 1990.

(9) Welch, J.; Greenbaum, D.; Mostame, S.; Aspuru-Guzik, A. Efficient quantum circuits for diagonal unitaries without ancillas. *New Journal of Physics* **2014**, *16*, 033040.

(10) Dan, X.; Geva, E.; Batista, V. S. Simulating Non-Markovian Quantum Dynamics on NISQ Computers Using the Hierarchical Equations of Motion. *Journal of Chemical Theory and Computation* **2025**, *21*, 1530–1546.

(11) Seneviratne, A.; Walters, P. L.; Wang, F. Exact Non-Markovian Quantum Dynamics on the NISQ Device Using Kraus Operators. *ACS Omega* **2024**, *9*, 9666–9675.

(12) Nielsen, M. A.; Chuang, I. L. *Quantum computation and quantum information*; American Mathematical Society, 2010.

(13) Levy, E.; Shalit, O. M. Dilation theory in finite dimensions: the possible, the impossible and the unknown. *Rocky Mountain Journal of Mathematics* **2014**, *44*, 203–221.

(14) Sweke, R.; Sinayskiy, I.; Bernard, D.; Petruccione, F. Universal simulation of Markovian open quantum systems. *Physical Review A* **2015**, *91*, 062308.

(15) Hu, Z.; Xia, R.; Kais, S. A quantum algorithm for evolving open quantum dynamics on quantum computing devices. *Scientific Reports* **2020**, *10*, 3301.

(16) Head-Marsden, K.; Krastanov, S.; Mazziotti, D. A.; Narang, P. Capturing non-Markovian dynamics on near-term quantum computers. *Physical Review Research* **2021**, *3*, 013182.

(17) Schlimgen, A. W.; Head-Marsden, K.; Sager-Smith, L. M.; Narang, P.; Mazziotti, D. A. Quantum state preparation and nonunitary evolution with diagonal operators. *Physical Review A* **2022**, *106*, 022414.

(18) Wang, Y.; Mulvihill, E.; Hu, Z.; Lyu, N.; Shivpuje, S.; Liu, Y.; Soley, M. B.; Geva, E.; Batista, V. S.; Kais, S. Simulating Open Quantum System Dynamics on NISQ Com-

puters with Generalized Quantum Master Equations. *Journal of Chemical Theory and Computation* **2023**, *19*, 4851–4862.

(19) Vatan, F.; Williams, C. P. Optimal Quantum Circuits for General Two-Qubit Gates. *Physical Review A* **2004**, *69*, 032315.

(20) Vidal, G.; Dawson, C. M. Universal Quantum Circuit for Two-Qubit Transformations with Three Controlled-NOT Gates. *Physical Review A* **2004**, *69*, 010301.

(21) Khatri, S.; LaRose, R.; Poremba, A.; Cincio, L.; Sornborger, A. T.; Coles, P. J. Quantum-Assisted Quantum Compiling. *Quantum* **2019**, *3*, 140.

(22) Jones, T.; Brown, A.; Bush, I.; Benjamin, S. C. QuEST and high performance simulation of quantum computers. *Scientific reports* **2019**, *9*, 10736.

(23) Cowtan, A.; Dilkes, S.; Duncan, R.; Krajenbrink, A.; Simmons, W.; Sivarajah, S. On the qubit routing problem. *arXiv preprint arXiv:1902.08091* **2019**,

(24) Murali, P.; Baker, J. M.; Javadi-Abhari, A.; Chong, F. T.; Martonosi, M. Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers. Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). 2019; pp 1015–1029.

(25) Barenco, A.; Bennett, C. H.; Cleve, R.; DiVincenzo, D. P.; Margolus, N.; Shor, P.; Sleator, T.; Smolin, J. A.; Weinfurter, H. Elementary gates for quantum computation. *Physical Review A* **1995**, *52*, 3457–3467.

(26) Cybenko, G. Reducing quantum computations to elementary unitary operations. *Computing in Science & Engineering* **2001**, *3*, 27–32.

(27) Krol, A. M.; Al-Ars, Z. Beyond Quantum Shannon: Circuit Construction for General n-Qubit Gates Based on Block ZXZ-Decomposition. 2024.

(28) Bullock, S. S.; Markov, I. L. Asymptotically Optimal Circuits for Arbitrary n-qubit Diagonal Computations. 2003.

(29) Mottonen, M.; Vartiainen, J. J. Decompositions of general quantum gates. 2005.

Supporting Information for

QFlux: Quantum Circuit Implementations of Molecular Dynamics.

Part III – State Initialization and Unitary Decomposition

Alexander V. Soudackov[†], Delmar G. A. Cabral[†], Brandon C. Allen[†], Xiaohan Dan[†], Nam P. Vu[†], Cameron Cianci[‡], Rishab Dutta[†], Sabre Kais[¶], Eitan Geva[§] and Victor S. Batista[*,‖,⊥]

[†]Department of Chemistry, Yale Quantum Institute, Yale University, New Haven, CT 06511, USA

[‡]Department of Physics, University of Connecticut, Storrs, CT 06268, USA

[¶]Department of Electrical and Computer Engineering, Department of Chemistry, North Carolina State University, Raleigh, North Carolina 27606, USA

[§]Department of Chemistry, University of Michigan, Ann Arbor, MI 48109, USA

[‖]Department of Chemistry, Yale University, New Haven, CT 06520, USA

[⊥]Yale Quantum Institute, Yale University, New Haven, CT 06511, USA

E-mail: `victor.batista@yale.edu`

Contents

S.1 State Preparation via Quantum Multiplexors

This section collects helper routines for mapping an arbitrary n-qubit state vector to the computational vacuum state $|0\cdots 0\rangle$ using a sequence of single-qubit rotations arranged in multiplexor form. The same building blocks can be inverted to construct state-preparation circuits.

S.1.1 Bloch Sphere Angles

This utility takes the two complex amplitudes (c_0, c_1) of a single qubit, normalizes them, and returns the equivalent Bloch-sphere angles θ and ϕ such that $\alpha |0\rangle + \beta |1\rangle = \cos(\theta/2) |0\rangle + \mathrm{e}^{\mathrm{i}\phi} \sin(\theta/2) |1\rangle$. These parameters are later used to build R_Z and R_Y rotations that map the qubit to $|0\rangle$ or $|1\rangle$.

Script S.1.1: Compute Bloch Sphere Angles

```python
import numpy as np
def compute_bloch_angles(c0, c1):
    norm = np.sqrt(np.abs(c0)**2 + np.abs(c1)**2)
    alpha, beta = c0/norm, c1/norm
    theta = 2 * np.arccos(np.abs(alpha))
    phi = np.angle(beta * np.conjugate(alpha) / (np.abs(beta)*np.abs(alpha)))
    return theta, phi
```

S.1.2 Single-Qubit Rotation Matrices

The following functions construct 2×2 rotation matrices $R_Z(\phi)$ and $R_Y(\theta)$ in the computational basis. Together with the angles from Section S.1.1, they generate the single-qubit unitaries used inside each 2×2 block of the multiplexor.

Script S.1.2: Rotation Matrices around Z and Y Axes

```python
def rz_matrix(phi):
    return np.array([[np.exp(-1j*phi/2), 0],
                     [0, np.exp(1j*phi/2)]])

def ry_matrix(theta):
    return np.array([[np.cos(theta/2), -np.sin(theta/2)],
                     [np.sin(theta/2),  np.cos(theta/2)]])
```

S.1.3 Block-Diagonal Multiplexor Construction

Given an n-qubit state vector of length 2^n, this routine builds a block-diagonal matrix whose 2×2 blocks act on adjacent amplitude pairs (c_0, c_1), choosing rotations that map each pair to either $|0\rangle$ or $|1\rangle$ depending on the flag `bit`. This is the basic "quantum multiplexor" primitive used in the recursive state-to-vacuum transformation.

Script S.1.3: Constructing the Multiplexor Matrix

```python
from scipy.linalg import block_diag

def multiplexor_matrix(n, vector, bit=0):
    bit = int(bool(bit))
    multiplexor = None
    for i in np.arange(0,2**n,2):
        c0, c1 = vector[i], vector[i+1]
        theta, phi = compute_bloch_angles(c0, c1)
        r = ry_matrix(bit*np.pi - theta) @ rz_matrix(-phi)
        multiplexor = block_diag(multiplexor, r) if multiplexor is not None else r
    return multiplexor
```

S.1.4 Recursive Transformation to the Vacuum State

The next routine repeatedly applies multiplexors on successively coarse-grained vectors to construct a global unitary that maps an arbitrary n-qubit state $|\psi\rangle$ to $|0 \cdots 0\rangle$. The final matrix `total_matrix` is the full $2^n \times 2^n$ unitary implementing this transformation, up to a global phase.

Script S.1.4: Recursive Quantum Multiplexor Transformation

```
def rotate_to_vacuum_matrix(vector_input):
   ndim = vector_input.size
   if ndim & (ndim - 1) != 0:
      raise ValueError("Dimension must be a power of 2.")
   n = int(np.log2(ndim))
   total_matrix = np.eye(ndim)
   vector_k = vector_input.copy()
   for k in np.arange(n,0,-1):
      vector_pruned = vector_k if k==n else vector_new[::2]
      multiplexor = multiplexor_matrix(k, vector_pruned)
      multiplexor_padded = multiplexor if k==n
                  else np.kron(multiplexor, np.eye(2**(n-k)))
      total_matrix = multiplexor_padded @ total_matrix
      vector_k = multiplexor_padded @ vector_k
      vector_new = multiplexor @ vector_pruned
   total_matrix *= np.conjugate(vector_k[0])
   return total_matrix
```

S.1.5 Example: Three-Qubit State Preparation

This example draws a random normalized 3-qubit state, constructs the corresponding “rotate-to-vacuum" unitary, and verifies that the forward transformation maps $|\psi\rangle$ to $|000\rangle$ while the Hermitian conjugate recovers the original state vector.

Script S.1.5: State preparation for a 3-qubit system

```
from numpy import linalg as LA
np.random.seed(42)
nq = 3
state_vector = (2*np.random.rand(8)-1) * np.exp(1j*2*np.pi*np.random.rand(8))
state_vector /= LA.norm(state_vector)
mrot = rotate_to_vacuum_matrix(state_vector)
rot_vector = mrot.dot(state_vector)
back_vector = np.conjugate(mrot.T).dot(rot_vector)
```

The forward transformation maps $|\psi\rangle \to |000\rangle$, and applying its Hermitian conjugate restores the original state, $|\psi\rangle = m_{\text{rot}}^{\dagger} |000\rangle$.

S.1.6 Example: Coherent Wavepacket State Preparation on a 6-Qubit System

This example initializes a coherent wavepacket state $|\psi\rangle$ in the position grid representation with $2^6 = 64$ points on a 6-qubit register, constructs the corresponding “rotate-to-vacuum" unitary `mrot`, and verifies that the forward transformation maps $|\psi\rangle$ to $|000000\rangle$ while the Hermitian conjugate recovers the original state vector.

Script S.1.6: Coherent Wavepacket State Preparation on a 6-Qubit System

```
import numpy as np
nq = 6

hbar = 1.0   # Planck's constant in atomic units
mass = 1.0   # mass in atomic units
omega = 1.0  # oscillator frequency
normalization = (mass*omega/np.pi/hbar)**(0.25)

# position grid
xmin = -5.0
xmax = 5.0
N_pts = 2**nq
dx = (xmax - xmin)/(N_pts-1)
xgrid = xmin + dx * np.arange(N_pts)

# coherent wavepacket localized at (x_0, p_0)
x_0 = 1.0
p_0 = 0.0
psi = np.sqrt(dx)*normalization*np.exp(-(mass*omega/hbar/2.0)*((xgrid-x_0)**2)
    + 1j*p_0*xgrid/hbar)

mrot = rotate_to_vacuum_matrix(psi)
vacuum_vector = mrot.dot(psi)
back_vector = np.conjugate(mrot.T).dot(rot_vector) # should be psi
```

The forward transformation maps $|\psi\rangle \rightarrow |000000\rangle$, and applying its Hermitian conjugate restores the original state, $|\psi\rangle = m_{\text{rot}}^{\dagger}\, |000\rangle$.

S.2 Givens-Based Decomposition of Unitaries

This section implements a Givens-rotation-based scheme to triangularize an $N \times N$ unitary matrix U using two-level unitaries that act on pairs of rows. The resulting product R is such that RU is upper triangular (and, after a phase adjustment, equal to the identity).

S.2.1 Imports and Setup

The first cell installs the `graycode` helper package and imports NumPy and Gray-code routines used to order the Givens sequence in a Gray-encoded basis.

Script S.2.1: Imports and Setup for Givens Decomposition

```
!pip install graycode
import numpy as np
import graycode
```

S.2.2 Single Givens Rotation

This function constructs a two-level unitary G that mixes rows j and k (for a fixed column i) to zero out the element $A_{j,i}$. It is the basic building block in the QR-like elimination process applied to a unitary matrix A.

Script S.2.2: Single Givens Rotation Constructor

```
def GivensRotation(i, j, k, A):
    """
    Construct a two-level unitary that zeros out A[j,i] by rotating rows j and k.
    Indices i, j, k are zero-based.
    """
    ndim = A.shape[0]
    G = np.eye(ndim, dtype=A.dtype)
    aji = A[j, i]
    if aji == 0:
        return G
    aki = A[k, i]
    norm = np.sqrt(abs(aji)**2 + abs(aki)**2)
    G[k, k] = np.conj(aki) / norm
    G[j, j] = aki / norm
    G[k, j] = np.conj(aji) / norm
    G[j, k] = -aji / norm
    return G
```

S.2.3 Full Givens Sequence and Triangularization

The routine `Gmatrix` sweeps column by column through a unitary U, applying a sequence of Givens rotations to eliminate subdiagonal entries. An initial global phase adjustment ensures that the final triangular matrix is proportional to the identity, and the product RU is close to the identity up to numerical precision.

Script S.2.3: Sequential Application of Givens Rotations

```
def Gmatrix(U, gray=False, print_sequence=False):
    """
    Apply a full sequence of Givens rotations to triangularize U.
    Returns R @ Uphase with a det-phase adjustment.
    """
    ndim = U.shape[0]
    n = int(np.log2(ndim))
    R = np.eye(ndim, dtype=U.dtype)

    # Global phase adjustment so the final triangular form is the identity.
    detU = np.linalg.det(U)
    phase = np.exp(-1j * np.angle(detU) / ndim)
    Uphase = phase * np.eye(ndim, dtype=U.dtype)
    Um = Uphase @ U

    # Perform eliminations column by column
    if print_sequence:
        print("U = D", end=' ')
    gray_list = list(graycode.gen_gray_codes(n)) if gray else None
    for i in range(ndim - 1):
        for j in range(ndim - 1, i, -1):
            if gray:
                ii = int(bin(gray_list[i]), 2)
                jj = int(bin(gray_list[j]), 2)
                kk = int(bin(gray_list[j-1]), 2)
            else:
                ii, jj, kk = i, j, j-1
            if print_sequence:
                print(f"G^+[{ii+1},{jj+1},{kk+1}]", end=' ')
            G = GivensRotation(ii, jj, kk, Um)
            R = G @ R
            Um = G @ Um

    if print_sequence:
        print("\nD = (", np.conjugate(phase), ")*I")

    return R @ Uphase
```

S.2.4 Example: Decomposition of a Random 4×4 Unitary

This test script generates a random 4×4 unitary matrix, applies the Givens-based decomposition with Gray ordering, and confirms that RU is numerically equal to the identity by printing the transformed matrix and its Frobenius norm deviation.

Script S.2.4: Decomposition of a Random [4×4] Unitary

```python
from scipy.stats import unitary_group
import numpy as np

def format_complex(z):
   return f"{z.real: .4f}{z.imag:+.4f}j"

def print_nicely(M):
   for row in M:
      print(" ".join(f"{format_complex(x):>16s}" for x in row))

# Reproducible random unitary
rng = np.random.default_rng(43)
ndim = 4
U = unitary_group.rvs(ndim, random_state=rng)

print("\nTarget unitary matrix U:")
print("------------------------")
print_nicely(U)

# Decomposition with Gray ordering
print("\nDecomposition:")
print("--------------")
RU = Gmatrix(U, gray=True, print_sequence=True)

U_transformed = RU @ U

print("\nTransformed matrix R @ U (should be identity):")
print("----------------------------------------------")
print_nicely(U_transformed)

# Numerical check
err = np.linalg.norm(U_transformed - np.eye(ndim))
print(f"\n||R @ U - I||_F = {err:.3e}")
assert np.allclose(U_transformed, np.eye(ndim), atol=1e-10)
```

The Givens sequence R is constructed so that RU is upper triangular and, after removing a global phase, numerically equal to the identity within floating-point precision.

S.3 Walsh Decomposition of Diagonal Unitaries

This section provides helper routines for decomposing diagonal unitaries into a product of Pauli-Z strings with single-qubit R_Z rotations, using a fast Walsh-Hadamard transform (FWHT) to extract expansion coefficients and a Gray-ordered circuit synthesis.

S.3.1 Binary and Gray-Code Utilities

These helper functions manipulate bit strings in MSB-first convention, compute Gray-code indices, and locate the most significant nonzero bit in a bit vector. They are used when organizing Walsh-string indices and assigning control/target qubits.

Script S.3.1: MSB-First Bits and Gray Code

```python
import numpy as np

def bits_msb(x, n):
   """MSB-first bit vector of length n."""
   return np.fromiter(f"{x:0{n}b}", dtype='U1').astype(int)

def gray_index(i):
   """Integer Gray code index: i ^ (i >> 1)."""
   return i ^ (i >> 1)

def msb_pos(bits):
   """Index (0..n-1, MSB-first) of highest-order 1, or -1 if all zero."""
   idx = np.where(bits == 1)[0]
   return -1 if idx.size == 0 else idx[-1]
```

S.3.2 Walsh Coefficients via FWHT

The following routines implement an in-place FWHT to map a vector of real phases f_k (associated with diagonal entries $\mathrm{e}^{\mathrm{i} f_k}$) to Walsh coefficients a_j. These coefficients parameterize the diagonal unitary as $\exp\Big(\mathrm{i}\sum_j a_j w_j\Big)$ in the Walsh basis.

Script S.3.2: Walsh Coefficients via FWHT

```python
def fwht_inplace(a):
    """In-place Fast Walsh-Hadamard Transform (Hadamard ordering)."""
    h = 1
    n = a.shape[0]
    while h < n:
        for i in range(0, n, h << 1):
            j_end = i + h
            for j in range(i, j_end):
                x = a[j]
                y = a[j + h]
                a[j]     = x + y
                a[j + h] = x - y
        h <<= 1
    return a

def walsh_coefficients_from_phases(f):
    """
    Given real phases f_k (length 2**n), return a_j per Eq. (\\ref{eq:walsh_coef}).
    """
    f = np.asarray(f, dtype=float).copy()
    a = fwht_inplace(f) / f.size
    return a

def phases_from_diagonal(U_diag):
    """
    Accept a 1D array of diagonal entries of U (|U_kk|=1) and return principal phases
     f_k.
    """
    U_diag = np.asarray(U_diag, dtype=complex)
    return np.angle(U_diag)  # principal branch (-pi, pi]
```

S.3.3 Walsh Strings and Gate List

Given the coefficients a_j, this function constructs a Gray-ordered list of CNOT and R_Z gates realizing the diagonal unitary $U = \exp\left(\mathrm{i} \sum_j a_j w_j\right)$. Each nonzero Walsh term is mapped to a parity network (CNOT ladder) feeding a single R_Z rotation on the highest-order nonzero qubit, followed by uncomputation.

Script S.3.3: Walsh Gate List (Gray-Ordered)

```python
def walsh_gate_list(a, n_qubits, eps=1e-9):
    """
    Produce a Gray-ordered list of ('C', c, t) CNOTs and ('R', q, theta) Rz for
     U=exp(i sum a_j w_j).
    Qubit indices use MSB-first order: q=0 is the most significant qubit.
    """
    assert len(a) == 2**n_qubits
    gates = []

    # Iterate Gray order over j=1..2^n-1 (the j=0 term is a global phase Rz on no
     qubit)
    for i in range(1, 2**n_qubits):
        j = gray_index(i)
        aj = a[j]
        if abs(aj) < eps:
            continue
        bits = bits_msb(j, n_qubits)          # MSB-first
        t = msb_pos(bits)                   # target = highest one bit
        if t < 0:                         # shouldn't happen for j>=1
            continue

        # forward parity ladder onto target
        for c in range(0, t):
            if bits[c] == 1:
                gates.append(('C', c, t))

        # the single-qubit Z rotation: \ee^{\ii a Z} = Rz(-2 a)
        gates.append(('R', t, -2.0 * aj))

        # uncompute parity
        for c in range(t-1, -1, -1):
            if bits[c] == 1:
                gates.append(('C', c, t))

    return gates
```

S.3.4 Circuit Construction and Peephole Optimization

The next helper cancels immediately repeated CNOTs and converts the gate list into a Qiskit `QuantumCircuit`. This simple peephole optimization reduces entangling depth in the synthesized diagonal circuit.

Script S.3.4: Build and Peephole-Optimize Circuit

```python
from qiskit import QuantumRegister, QuantumCircuit

def cx_cancel(gates):
   """Cancel adjacent identical CNOTs."""
   out = []
   for g in gates:
      if out and g[0]=='C' and g == out[-1]:
         out.pop()
      else:
         out.append(g)
   return out

def build_qiskit(gates, n_qubits):
   qr = QuantumRegister(n_qubits, 'q')
   qc = QuantumCircuit(qr)
   for tag,*args in gates:
      if tag == 'C':
         c, t = args
         qc.cx(c, t)
      elif tag == 'R':
         q, theta = args
         qc.rz(theta, q)
      else:
         raise ValueError(f"Unknown gate {tag}")
   return qc
```

S.3.5 Walsh Decomposition Circuit End-to-End Test

This script starts from a random diagonal unitary $U = \mathrm{diag}(e^{if_k})$, computes a_j via FWHT, generates and optimizes the Gray-ordered circuit, and prints its depth.

Script S.3.5: Build and Peephole-Optimize Circuit

```python
# Example: 3-qubit diagonal unitary
n = 3
rng = np.random.default_rng(7)
f = rng.uniform(-np.pi, np.pi, size=2**n)  # phases f_k
a = walsh_coefficients_from_phases(f)     # a_j via FWHT

# Gate list -> small peephole optimization -> circuit
gates = walsh_gate_list(a, n)
```

```
gates_opt = cx_cancel(gates)
qc = build_qiskit(gates_opt, n)

print("Walsh coefficients a_j:", np.round(a, 6))
print("\nOptimized circuit:")
print(qc)
print("\nCircuit depth:", qc.depth(), "  # quick proxy for entangling depth")
```

S.3.6 Solution: Walsh Synthesis for a Double-Well Potential

Setup. We consider a single nuclear degree of freedom x evolving on a symmetric double-well potential

$$V(x) = V_0 \left(\frac{x^2}{x_0^2} - 1 \right)^2, \tag{S.1}$$

with barrier height V_0 and minima at $x = \pm x_0$.

The coordinate is discretized on a uniform grid encoded by $n = 4$ qubits,

$$x_j = x_{\min} + j\,\Delta x, \qquad j = 0, 1, \dots, 2^n - 1, \tag{S.2}$$

where $\Delta x = (x_{\max} - x_{\min})/(2^n - 1)$. In the computational basis $\{|j\rangle\}$, the potential-energy operator is diagonal,

$$V = \sum_{j=0}^{2^n-1} V(x_j)\ |j\rangle\langle j|\,. \tag{S.3}$$

The corresponding time-evolution operator is

$$U_V(t) = \mathrm{e}^{-\mathrm{i}Vt}. \tag{S.4}$$

Walsh (Pauli-Z) expansion. Any diagonal operator on n qubits admits an exact expansion in the Walsh (Pauli-Z) basis,

$$H_V = \sum_{S \subseteq \{1,\dots,n\}} c_S \prod_{k \in S} Z_k, \tag{S.5}$$

where the coefficients c_S are obtained from the Walsh–Hadamard transform of the sampled values $\{V(x_j)\}$.

For $n = 4$, this expansion contains at most $2^4 = 16$ diagonal Pauli strings,

$$V = c_\emptyset I + \sum_i c_i Z_i + \sum_{i<j} c_{ij} Z_i Z_j + \sum_{i<j<k} c_{ijk} Z_i Z_j Z_k + c_{1234} Z_1 Z_2 Z_3 Z_4. \tag{S.6}$$

The identity term contributes only a global phase and is omitted in what follows.

Factorized time evolution. Because all Pauli-Z strings commute, the potential-energy propagator factorizes exactly:

$$U_V(t) = \prod_{S \neq \emptyset} \exp\left(-\mathrm{i} t \, c_S \prod_{k \in S} Z_k \right). \tag{S.7}$$

As a result, $U_V(t)$ can be implemented without Trotter error.

Circuit cost per term. A unitary of the form

$$\exp(-\mathrm{i}\theta Z_{i_1} Z_{i_2} \cdots Z_{i_m})$$

is implemented by computing the parity of the m qubits onto a single target qubit, applying an $R_z(2\theta)$ rotation, and uncomputing the parity. This requires $2(m-1)$ CNOT gates and one single-qubit R_z rotation (Part II, Sec. 3). Therefore:

- $m = 1$ (single-qubit terms): 0 CNOTs,
- $m = 2$ (two-qubit terms): 2 CNOTs,
- $m = 3$ (three-qubit terms): 4 CNOTs,
- $m = 4$ (four-qubit term): 6 CNOTs.

Total CNOT count. If the Walsh expansion contains N_1 single-qubit terms, N_2 two-qubit terms, N_3 three-qubit terms, and N_4 four-qubit terms, the total number of CNOT gates is

$$N_{\text{CNOT}} = 2N_2 + 4N_3 + 6N_4. \tag{S.8}$$

The total number of single-qubit rotations is

$$N_{R_z} = N_1 + N_2 + N_3 + N_4. \tag{S.9}$$

Comparison to generic synthesis. A fully generic four-qubit unitary requires on the order of $\mathcal{O}(4^n) \sim 400$ CNOT gates, while even a generic four-qubit diagonal unitary requires several tens to hundreds of entangling gates (105–536 CNOT gates, as shown in Table 2). In contrast, according to Eq. (S.8), the Walsh-based synthesis requires only 56 CNOT gates for this problem and scales with the number of nonzero Walsh coefficients, which for smooth potential-energy surfaces such as the double-well is typically far smaller than 2^n.

Physical interpretation. The resulting circuit implements the exact phase accumulation

$$|x_j\rangle \longrightarrow \mathrm{e}^{-\mathrm{i}V(x_j)t}\,|x_j\rangle\,,$$

corresponding to the potential-energy step in split-operator quantum dynamics, as discussed in Part II (Sec. 5). When alternated with the kinetic-energy propagator, this enables simulation of tunneling, vibrational level splitting, and coherent wavepacket transfer between the two wells.

Conclusion. This solution illustrates how Walsh-based synthesis exploits the diagonal and chemically structured nature of discretized potential energy surfaces to achieve dramatic reductions in entangling-gate counts. Such structure-aware compilation is essential for scalable quantum simulations of molecular dynamics on NISQ-era and early fault-tolerant devices.