Flux

QFlux: An Open-Source Python Package for Quantum Dynamics Simulations

QFlux: Quantum Circuit Implementations of Molecular Dynamics

Victor S Batista

Yale University, Department of Chemistry and Yale Quantum Institute

Part III: The QFlux Synthesis Pipeline

Every quantum algorithm depends on two operations:
preparing a state and implementing a unitary.
These are not physical problems — they're synthesis problems.
How linear algebra is compiled as pulses applied to quatum hardware

https://qflux.batistalab.com

Part_III.ipynb

QFlux: Quantum Circuit Implementations of Molecular Dynamics

Victor S Batista

Yale University, Department of Chemistry and Yale Quantum Institute

Part III: The QFlux Synthesis Pipeline

This tutorial is based on the manuscript

QFlux: Quantum Circuit Implementations for Molecular Dynamics

Part III – State Initialization and Unitary Decomposition

Authors:

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

QFlux: An Open-Source Python Package for Quantum Dynamics Simulations

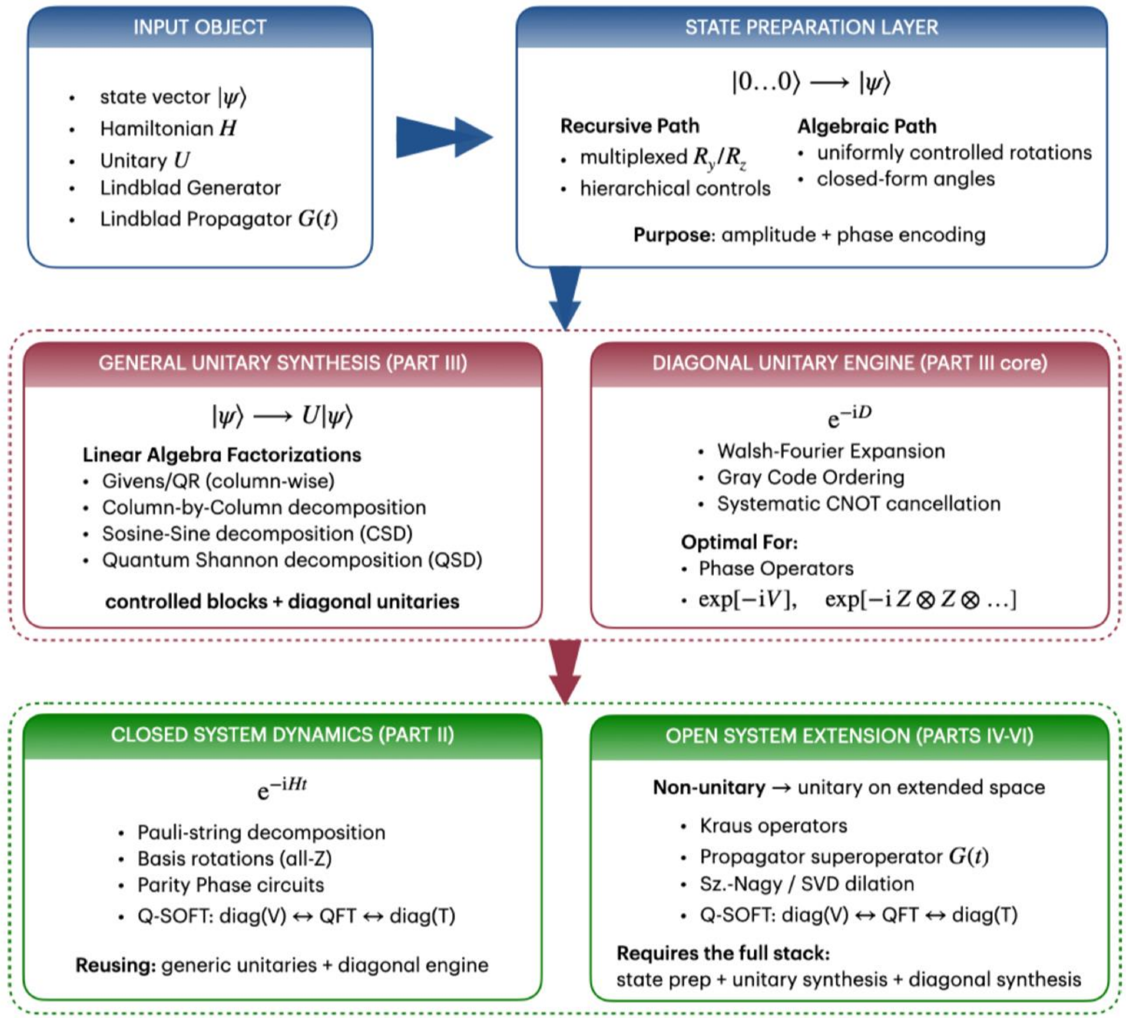
The QFlux Synthesis Pipeline

The backbone of QFlux

One pipeline for all simulations

- State vectors → state preparation
- Generic unitaries → linear-algebra factorizations
- Diagonal operators → Walsh synthesis

Reused across closed and open systems



Method Selection: Which Tool When?

State preparation:

- Multiplexors → transparent, good for debugging
- UCRs → compact, compiler-friendly

Unitaries:

- Givens / QR → pedagogical baseline
- Column-by-column → practical
- CSD / QSD → asymptotically optimal

Diagonal:

- Walsh + Gray code → NISQ-optimal

PART A — STATE PREPARATION

- **Input:** $$|00\cdots 0\rangle = |0\rangle^{\otimes n} = \underbrace{\begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} \otimes \cdots \otimes \begin{pmatrix}1\\0\end{pmatrix}}_{n\ \text{qubits}} = \boxed{(1,0,\ldots,0)^\top}$$

- **Target:** $$|\psi\rangle = \sum_{k_1,k_2,\ldots,k_n \in \{0,1\}} c_{k_1 k_2 \ldots k_n} |k_1 k_2 \ldots k_n\rangle$$
$$|\psi\rangle = \boxed{(c_1, c_2, \ldots, c_{2^n})^\top}$$

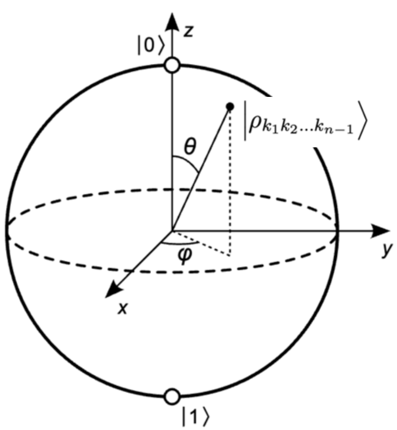
- **Goal:** deterministic, exact preparation

- **Strategy:** ***disentangle qubits one by one***

PART A — STATE PREPARATION

$$|\psi\rangle = \sum_{k_1,k_2,\dots,k_n \in \{0,1\}} c_{k_1k_2\dots k_n} |k_1k_2\dots k_n\rangle$$

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



$$\left|\rho_{k_1k_2\dots k_{n-1}}\right\rangle = c_{k_1k_2\dots k_{n-1}0}|0\rangle + c_{k_1k_2\dots k_{n-1}1}|1\rangle$$

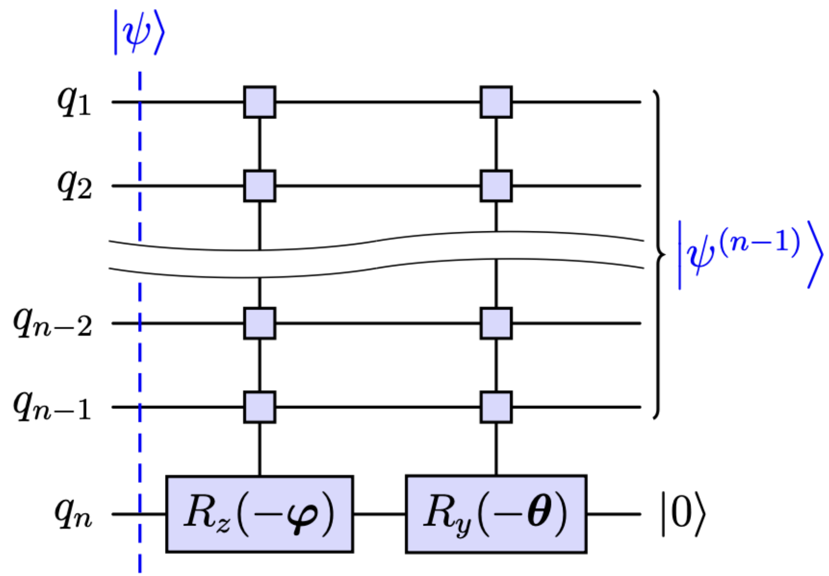
$$R_y(-\theta_{k_1k_2\dots k_{n-1}})\,R_z(-\varphi_{k_1k_2\dots k_{n-1}})\left|\rho_{k_1k_2\dots k_{n-1}}\right\rangle = r_{k_1k_2\dots k_{n-1}}\mathrm{e}^{\mathrm{i}t_{k_1k_2\dots k_{n-1}}}|0\rangle$$

Multiplexor Intuition

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

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

- Group amplitudes by last qubit
- Each conditional single-qubit state lies on Bloch sphere
- Rotate each to $|0\rangle$ using conditional rotations
- Remove entanglement one qubit at a time



What Is a Multiplexor?

For each bitstring of the control qubits, apply a different Ry–Rz pair to the target

- A gate that applies different single-qubit rotations depending on controls
- Block-diagonal structure:

    - One rotation per control configuration

- Acts as “conditional disentangler”

Script S.1.3: Part_III.ipynb

```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
```

Recursive Disentanglement Algorithm

- Step 1: rotate last qubit → $|0\rangle$
- Step 2: recurse on remaining n−1 qubits
- Reverse the sequence to prepare $|\psi\rangle$
- Cost: ~2^n CNOTs

Script S.1.5: [Part_III.ipynb]
Example: Three-Qubit State Preparation

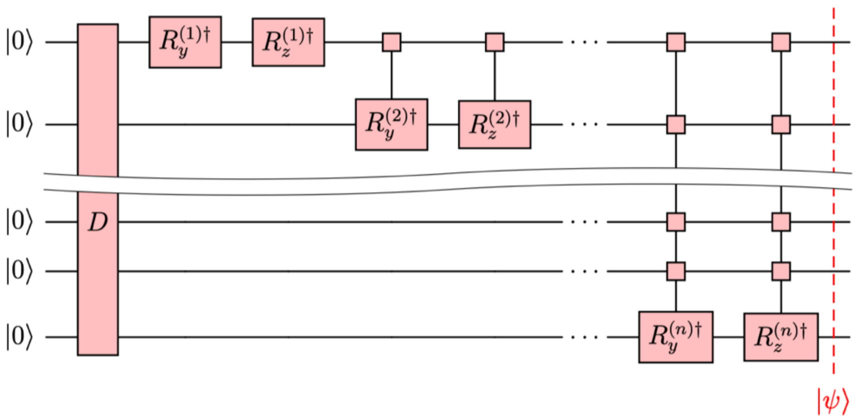
Script S.1.6: [Part_III.ipynb]
Example: Coherent Wavepacket State Preparation on a 6-Qubit System

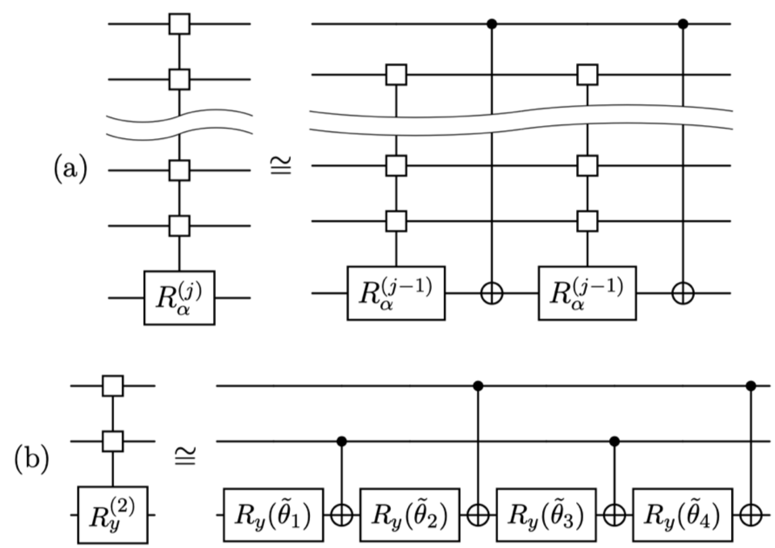
Script S.1.5: Example: Three-Qubit State Preparation

```python
from numpy import linalg as LA
#np.random.seed(42)
nq = 4
ndim = 2**nq
state_vector = (2*np.random.rand(ndim)-1) * np.exp(1j*2*np.pi*np.random.rand(ndim))
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)
```

Recursive Disentanglement Algorithm

Script S.1.4: Recursive Quantum Multiplexor Transformation





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

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

When to Use Multiplexors

- **Best for:**
  - Conceptual clarity
  - Debugging pipelines
  - Small n
- **Tradeoff:**
  - Higher CNOT count
  - Recursive, irregular structure

PART B — UNIFORMLY CONTROLLED ROTATIONS (UCR)

Motivation for UCRs

- Same goal: prepare arbitrary

$$|\psi\rangle = \left(|c_1|\mathrm{e}^{\mathrm{i}\omega_1}, |c_2|\mathrm{e}^{\mathrm{i}\omega_2}, \ldots, |c_N|\mathrm{e}^{\mathrm{i}\omega_N}\right)^\top$$

- But:
    - Non-recursive
    - Regular layered structure
    - Predictable gate counts

Key Idea: Separate Phase and Amplitude

First, we make the state real and positive

- **Step 1:** remove all relative phases with uniformly controlled z-rotations $\mathcal{R}_{\mathbf{z}}$

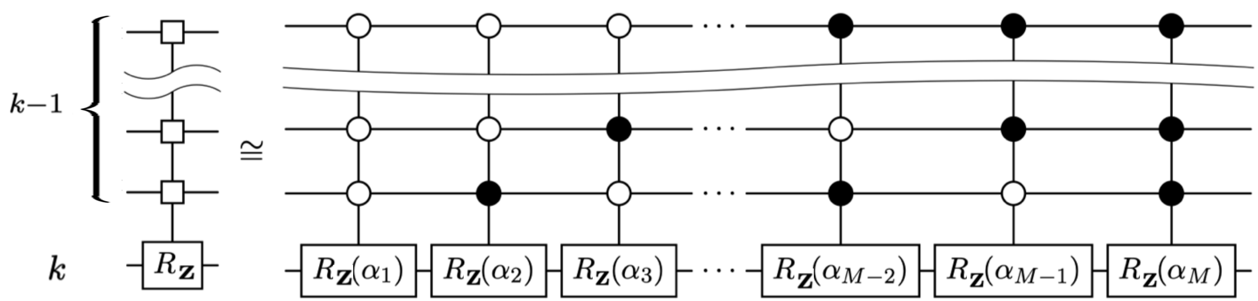
$$|\psi\rangle = \left(|c_1|\mathrm{e}^{\mathrm{i}\omega_1}, |c_2|\mathrm{e}^{\mathrm{i}\omega_2}, \ldots, |c_N|\mathrm{e}^{\mathrm{i}\omega_N}\right)^{\top}$$

Angles for phase equalization:

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

$$\mathcal{R}_{\mathbf{z}}\,|\psi\rangle = \mathrm{e}^{\mathrm{i}\Omega}(|c_1|, |c_2|, \ldots, |c_N|)^{\top} \qquad \mathcal{R}_{\mathbf{z}} = \prod_{k=1} F_k^{k-1}(\mathbf{z}, \boldsymbol{\alpha}^z_{n-k+1}) \otimes I_{2^{n-k}}$$

$F_k^{k-1}(\mathbf{z}, \boldsymbol{\alpha}^z_{n-k+1})$



Key Idea: Separate Phase and Amplitude

Second, we move probability mass across basis states.

- **Step 2:** redistribute amplitudes with uniformly controlled y-rotations $\mathcal{R}_{\mathbf{y}}$

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

Angles for cancelling amplitudes: (with the least significant qubit set to 1)

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

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

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

What is the asymptotic cost?

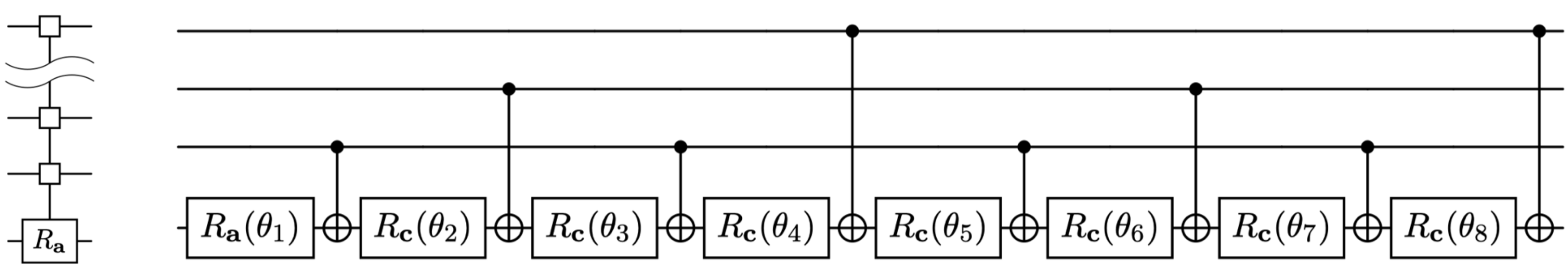
CNOT gates: $2^{n+2} - 4n - 4$; Single-qubit rotations: $2^{n+2} - 5$

Gray Code

$$M_{ij} = 2^{-k}(-1)^{b_{j-1}\cdot g_{i-1}}$$

- Control patterns reordered by Gray code
- Consecutive gates differ by one control bit
- Enables:
  - Minimal CNOTs
  - Systematic cancellations

$$\begin{bmatrix} \theta_1 \\ \vdots \\ \theta_{2^k} \end{bmatrix} = \mathbf{M} \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_{2^k} \end{bmatrix}$$



When to Use UCRs

In practice, this is your default state-preparation engine
Clean, compact, and hardware-friendly

- Best for:
  - Scalable synthesis
  - Automation
  - Compiler pipelines
- Cost:
  - Fewer CNOTs than multiplexors
  - Analytic angles

PART C — GENERIC UNITARY DECOMPOSITION

Problem: Arbitrary Unitary Synthesis

- Input: $U \in SU(2^n)$
- Goal: decompose into 1- and 2-qubit gates
- Constraint: minimize CNOT count

Givens Rotations (QR Intuition)

- Classical QR → eliminate off-diagonal entries column by column
- Quantum version:
  - Two-level rotations
  - Cancel one element at a time

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

$$ {}^{i}G_{j,k} = \begin{array}{c} \begin{array}{ccccc} & k & & j & \\ & \downarrow & & \downarrow & \end{array} \\ \begin{pmatrix} \ddots 1 \ddots & & & & \\ & U^*_{ki} & & U^*_{ji} & \\ & & \ddots 1 \ddots & & \\ & -U_{ji} & & U_{ki} & \\ & & & & \ddots 1 \ddots \end{pmatrix} \end{array} \begin{array}{c} \\ \\ \leftarrow k \\ \\ \leftarrow j \\ \\ \end{array} $$

QFlux: An Open-Source Python Package for Quantum Dynamics Simulations

Givens Rotations (QR Intuition)

Example: [4×4] unitary matrix U

$$U \xrightarrow{e^{-i\arg(\det U)/4} I} \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix} \xrightarrow{{}^1G_{4,3}} \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ 0 & * & * & * \end{pmatrix} \xrightarrow{{}^1G_{3,2}} \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix} \xrightarrow{{}^1G_{2,1}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix}$$

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

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

Gray Code Makes It Implementable

- Reorder basis states by Gray code
- Each rotation couples states differing by one qubit
- Implemented as multi-controlled single-qubit gates

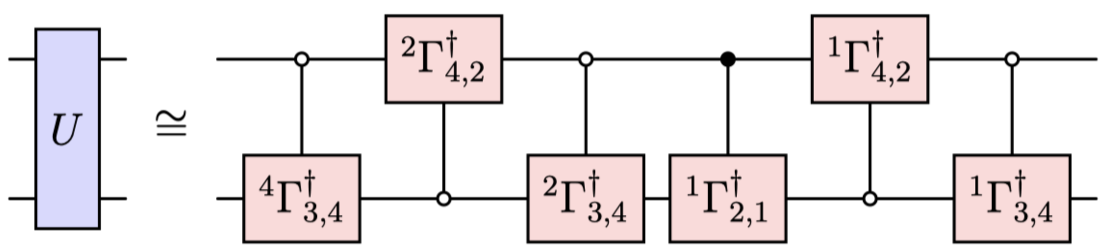
General formula:

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

$\gamma(\cdot)$ = Gray-code permutation (*i.e.*, the integer value of a Gray-coded bitstring)

Example: 2 qubit unitary

Part_III.ipynb **(Scripts S.2.1–S.2.4)**



Column-by-Column Decomposition (CCD)

State preparation on every column

- Apply reverse state preparation to each column
- Protect previously fixed columns with extra controls

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

with $D = \mathcal{R}_{2^n-1}^\dagger$

1. Rotate its first column to $|0\rangle$ (state preparation):

$$\mathcal{R}_0 U_0 |0\rangle = |0\rangle \quad \text{with} \quad U_0 := U$$

2. Rotate the second column of $U_1 := \mathcal{R}_0 U_0$ to $|1\rangle$:

$$\mathcal{R}_1 U_1 |1\rangle = |1\rangle \quad \text{with} \quad \mathcal{R}_1 |0\rangle = |0\rangle$$

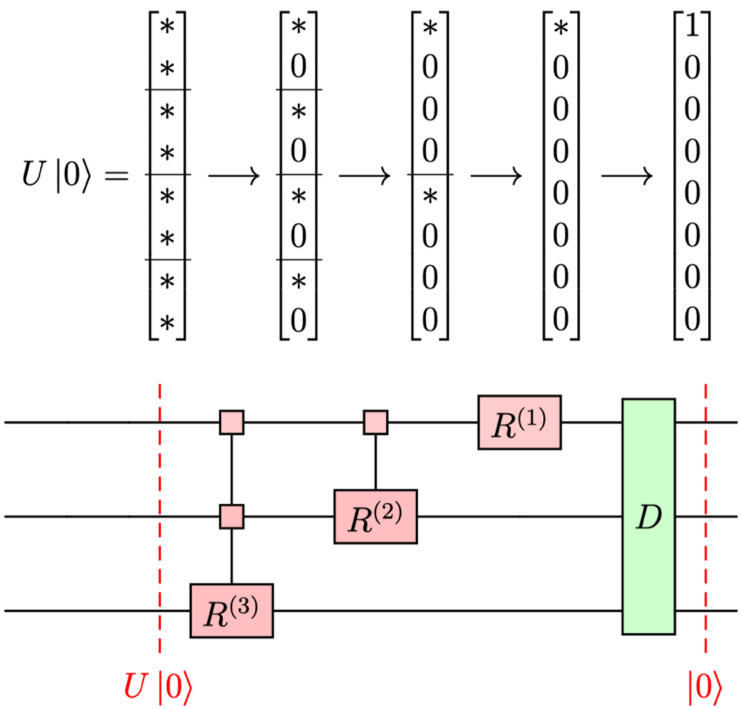
3. Rotate subsequent columns analogously:

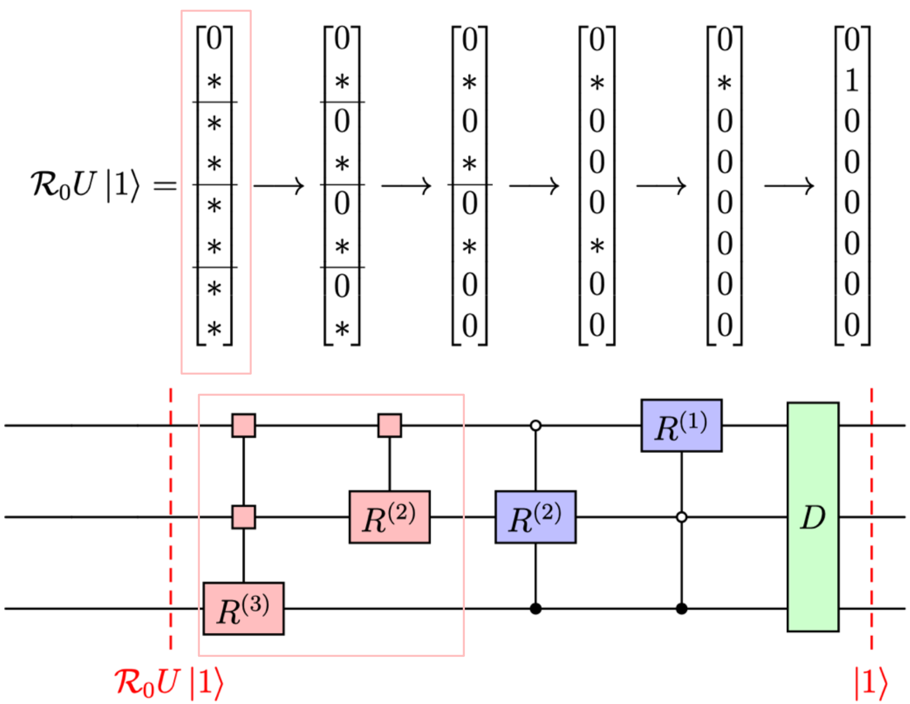
$$\mathcal{R}_j U_j |j\rangle = |j\rangle \quad \text{with} \quad U_{j+1} := \mathcal{R}_j U_j$$

$$\mathcal{R}_j |i\rangle = |i\rangle \text{ for all } i < j.$$

Column-by-Column Decomposition (CCD)

State preparation on every column





CSD & Quantum Shannon Decomposition

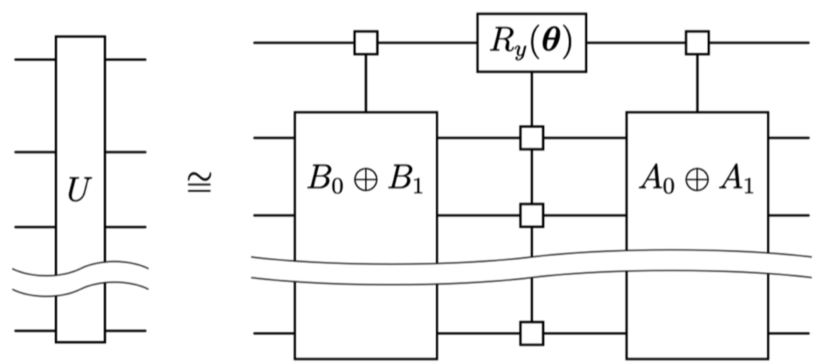
- Recursively split unitary into blocks
- Push all control into single-qubit multiplexed rotations

Cosine-Sine Decomposition (CSD):

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

$$C = \mathrm{diag}(\cos\tfrac{\theta_1}{2}, \ldots, \cos\tfrac{\theta_{2n-1}}{2})$$

$$S = \mathrm{diag}(\sin\tfrac{\theta_1}{2}, \ldots, \sin\tfrac{\theta_{2n-1}}{2})$$



Applying CSD recursively to $A_0 \oplus A_1$ and $B_0 \oplus B_1$

CSD & Quantum Shannon Decomposition

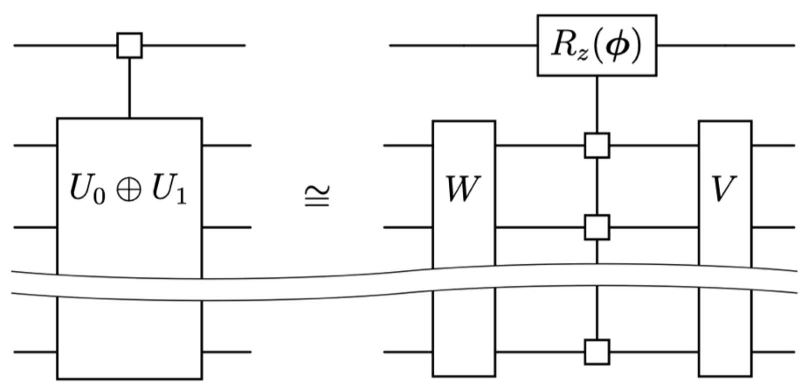
- Recursively split unitary into blocks
- Push all control into single-qubit multiplexed rotations

Quantum Shannon Decomposition (QSD):

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

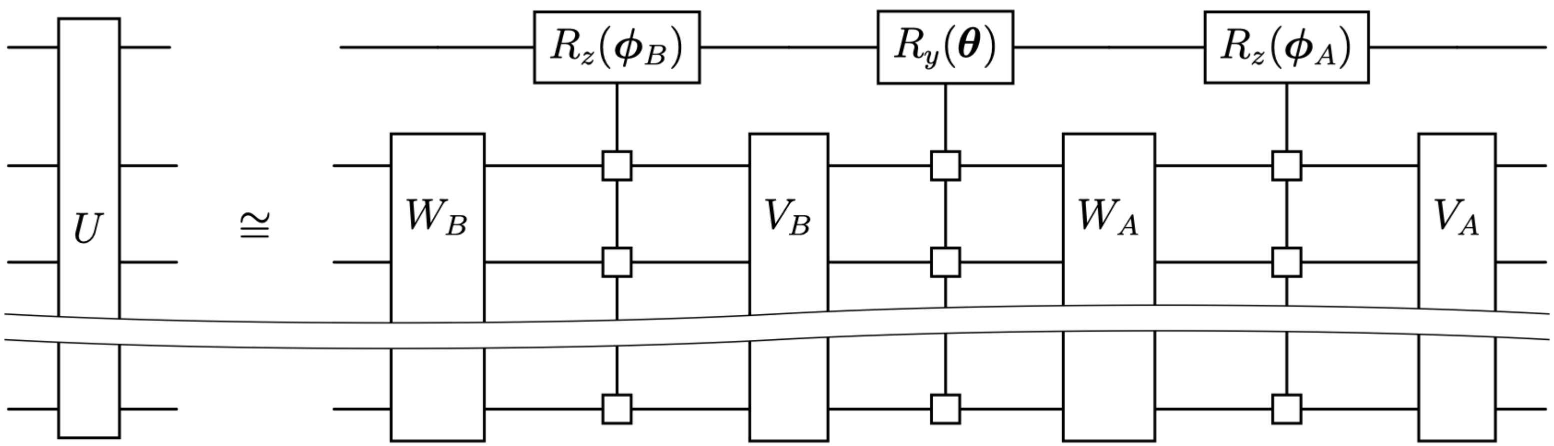
$$U = \left(\begin{array}{c|c} U_0 & 0 \\ \hline 0 & U_1 \end{array}\right)$$

The middle block $D \oplus D^\dagger$ is a diagonal multiplexor and maps to a single multiplexed $R_z(\phi)$ on the most significant qubit



CSD + Quantum Shannon Decomposition

- Recursively split unitary into blocks
- Push all control into single-qubit multiplexed rotations



What is the asymptotic cost? CNOT gates: CSD: $O(4^n)$ (smaller constants)

Optimized QSD: $\frac{23}{48}4^n - \frac{3}{2}2^n + O(1)$

PART D — DIAGONAL UNITARIES

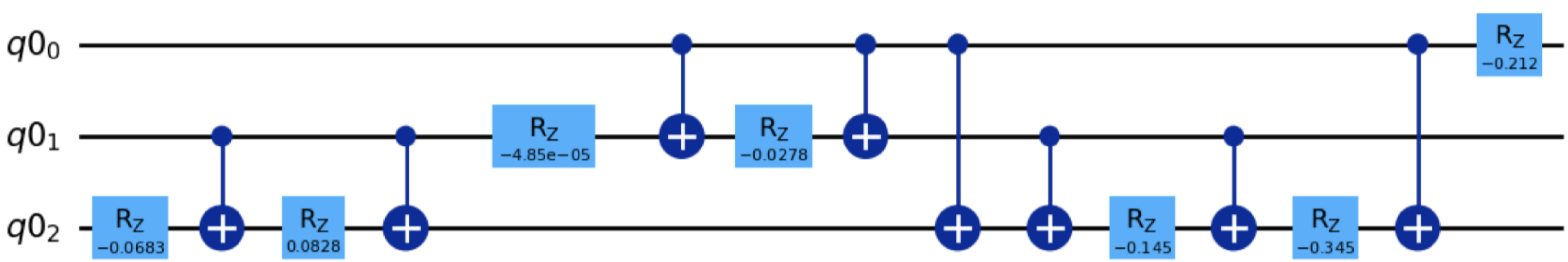
Special Case: Diagonal Operators

- Arise in:
  - e^{-iHt}
  - Phase oracles
  - Lindblad dilation

Best method: Walsh decomposition

Part_III.ipynb **Section S.3.6:** Walsh Synthesis of $\mathrm{e}^{-\mathrm{i}V(x_j)t}$ for a unitary of a Double-Well Potential

Scripts S.3.1-5: $U := \mathrm{diag}(\mathrm{e}^{\mathrm{i}f_k})$



Walsh + Gray Code = NISQ-Optimal

- Expand diagonal operator in Walsh basis
- Order terms by Gray code
- Systematic CNOT cancellation
- Cost: $O(2^n)$ entangling gates

$$U = \mathrm{e}^{\mathrm{i}F}$$

$$F = \mathrm{diag}(f_0, \ldots, f_{2^n-1}) \quad \text{real}$$

$$F = \sum_{j=0}^{2^n-1} a_j\, w_j$$

$$w_j := Z_1^{j_1} \otimes Z_2^{j_2} \otimes \cdots \otimes Z_n^{j_n}$$

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

$$a_j = 2^{-n}\,\mathrm{Tr}(w_j F)$$

Each factor $\mathrm{e}^{\mathrm{i}a_j w_j}$ *is efficiently realized with one single-qubit Z-rotation and at most 2n CNOTs*
Ordering terms by a Gray code cut two-qubit cost from naive $O(n2^n)$ *to roughly* $O(2^n)$ *in practice*

$$j = (j_1 \ldots j_n)_2 \text{ is the } \textit{binary} \text{ label}$$

$$j = \textstyle\sum_{l=1}^{n} j_l 2^{n-l}$$

Take-Home Messages

- State prep and unitary synthesis are the backbone of all simulations
- Operator structure determines optimal circuits
- QFlux provides:
  - Transparent algorithms
  - Predictable scaling
  - Reusable infrastructure