LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-821599

IBM Open Science Price - SWAP Gate Challenge

N. A. Petersson, F. Garcia, S. Guenther, Y. Choi

April 15, 2021

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

IBM Open Science Price – SWAP Gate Challenge

N. Anders Petersson* Fortino Garcia† Stefanie Günther*
Youngsoo Choi*

April 14, 2021

1 Introduction

On November 30, 2020, IBM announced an open science price competition aimed at improving the fidelity of a SWAP gate on their 7 qubit Quantum system named "Casablanca". The details of the SWAP gate problem was outlined in a Jupyter notebook, which utilizes the IBM developed open source package Qiskit. The participants were allowed to work in teams with up to five people and restricted to only use open source software in solving the problem. Their results were due on April 16, 2021.

Our approach to solving the SWAP gate challenge is based on the quantum optimal control techniques implemented in our open source packages Juqbox.jl and Quandary. The results from the optimal control approach is only as good as the accuracy in the description of the quantum system dynamics, characterized by a Hamiltonian model that was provided by IBM. Based on the calibrated control pulses IBM provide for the standard gate set, we developed a reverse engineering approach to calibrate our computational model. Techniques were developed to translate between Qiskit's pulse representation and the B-spline formulation used in Juqbox and Quandary. The fidelity of the optimized pulse sequences were estimated using Qiskit's randomized bench marking techniques.

2 Hamiltonian model

The Hamiltonian model use in this study is based on the paper by Magensan and Gambetta [3]. They considered a system of two transmons coupled by a bus resonator. The bus resonator is modeled as a harmonic oscillator with fundamental frequency ω^r and each transmon is coupled to the bus resonator by a Jaynes-Cummings Hamiltonian with coupling strength g_j. Let the 01 transition frequencies of the transmons be ω_j. It is assumed that the coupling is in the dispersive regime, which means that resonator frequency is sufficiently detuned from the 01 transition frequencies to make $|\omega_j - \omega^r| \gg |g_j|$.

After transforming the Hamiltonian to a frame rotating with frequency ω^r in all three subsystems (two transmons and a bus resonator), the system and control Hamiltonians become (for

*Lawrence Livermore National Laboratory, Livermore, CA, {petersson1, guenther5, choi15}@llnl.gov
†University of Colorado at Boulder, Boulder, CO, fortino.garcia@colorado.edu

notational convenience we set $\hbar = 1$)

$$H_{sys} = \sum_{j=1}^{2} \left((\omega_j - \omega^r)\, b_j^\dagger b_j + \frac{\Delta_j}{2} b_j^\dagger b_j^\dagger b_j b_j \right) + g_j \left(b_j^\dagger c + b_j c^\dagger \right), \tag{1}$$

$$H_{ctrl}(t) = \sum_{j=1}^{2} \mathrm{Re}(e^{i\omega^r t} d_j(t)) \left(e^{-i\omega^r t} b_j + e^{i\omega^r t} b_j^\dagger \right), \tag{2}$$

where b_j is the lowering operator for the j-th transmon and c is the lowering operator for the bus resonator. The above model would have been the idea starting point for an optimal control approach for designing a SWAP gate. Unfortunately, the resonator frequency ω^r and the coupling coefficients g_j are not readily available from the description of the Casablanca system.

Magensan and Gambetta [3] proceeds by deriving a simplified Hamiltonian model using the following steps: 1) reorder the state vector into blocks of increasing transmon excitation number, 2) adiabatically eliminate the terms that couple the blocks, 3) project onto the zero-excitation subspace of the bus resonator. These steps results in an effective Hamiltonian for the two transmon system given in Equation (2.12) of [3]. Because some of these steps are not described in detail, it has been difficult to assess the accuracy of the resulting Hamiltonian model. Nevertheless, this model is completely specified for the Casablanca system as all parameters can be accessed through the Qiskit interface. For these reasons, it is used as a starting point in our modeling effort. Based on this model, the lab frame system Hamiltonian for qubits 5 and 6 of Casablanca is

$$H_{sys} = \sum_{j=5}^{6} \left(\tilde{\omega}_j\, b_j^\dagger b_j + \frac{\Delta_j}{2} b_j^\dagger b_j^\dagger b_j b_j \right) + j_{56} \left(b_5^\dagger b_6 + b_5 b_6^\dagger \right). \tag{3}$$

Here $\tilde{\omega}_j$ is the dressed frequency of the j-th transmon, $b_5 = I \otimes a$, $b_6 = a \otimes I$ and a is the lowering matrix for a single system. In the following we drop the tilde on ω. Based on Equation (2.14) in [3] and the Hamiltonian entry in the backend for the Casablanca system, the lab frame control Hamiltonian satisfies

$$H_{ctrl}(t) = \Omega_{d,5} \left(D_5(t) + U_{10}^{(5,6)}(t) \right) (b_5 + b_5^\dagger) + \Omega_{d,6} \left(D_6(t) + U_{11}^{(6,5)}(t) \right) (b_6 + b_6^\dagger), \tag{4}$$

where

$$D_5(t) = \mathrm{Re}\left(e^{i\omega_5 t} d_5(t) \right), \quad U_{10}^{(5,6)}(t) = \mathrm{Re}\left(e^{i\omega_6 t} u_{10}(t) \right), \tag{5}$$

$$D_6(t) = \mathrm{Re}\left(e^{i\omega_6 t} d_6(t) \right), \quad U_{11}^{(5,6)}(t) = \mathrm{Re}\left(e^{i\omega_5 t} u_{11}(t) \right). \tag{6}$$

For conciseness we have absorbed the phase factors $e^{i\phi}$ into the normalized control functions d_5 through u_{11}. Note that U_{10} is applied to qubit 5 but uses qubit 6's transition frequency. Correspondingly, U_{11} is applied to qubit 6, but uses qubit 5's transition frequency. Because $\mathrm{Re}(z) = 0.5(z + \bar{z})$ for $z \in \mathbb{C}$, the control Hamiltonian can also be written

$$\begin{aligned} H_c(t) =& \frac{1}{2} \Omega_{d,5} \left(e^{i\omega_5 t} d_5(t) + e^{-i\omega_5 t} \bar{d}_5(t) + e^{i\omega_6 t} u_{10}(t) + e^{-i\omega_6 t} \bar{u}_{10}(t) \right) (b_5 + b_5^\dagger) \\ &+ \frac{1}{2} \Omega_{d,6} \left(e^{i\omega_6 t} d_6(t) + e^{-i\omega_6 t} \bar{d}_6(t) + e^{i\omega_5 t} u_{11}(t) + e^{-i\omega_5 t} \bar{u}_{11}(t) \right) (b_6 + b_6^\dagger) \end{aligned} \tag{7}$$

We apply a rotating frame transformation using the same frequency of rotation for both subsystems (e.g. $\omega_{rot} = \omega_5$). In this frame, the system Hamiltonian becomes

$$\widetilde{H}_{sys} = R H_{sys} R^\dagger + i R^\dagger \dot{R} \tag{8}$$

$$= \sum_{j=5}^{6} \left(\delta_j\, b_j^\dagger b_j + \frac{\Delta_j}{2} b_j^\dagger b_j^\dagger b_j b_j \right) + j_{56} \left(b_5^\dagger b_6 + b_5 b_6^\dagger \right) \tag{9}$$

where we have defined $\delta_j = \omega_j - \omega_{rot}$ for $j = 5, 6$. After applying the rotating wave approximation, the control Hamiltonian becomes

$$\widetilde{H}_c(t) \approx \frac{\Omega_{d,5}}{2}\left(e^{i\delta_5 t} d_5(t) + e^{i\delta_6 t} u_{10}(t)\right) b_5 + \frac{\Omega_{d,6}}{2}\left(e^{i\delta_6 t} d_6(t) + e^{i\delta_5 t} u_{11}(t)\right) b_6 + \text{H.c.}, \tag{10}$$

where H.c. stands for the Hermitian conjugate.

3 Optimal control with Juqbox.jl and Quandary

We use numerical optimization in the open source packages Juqbox.jl [1] and Quandary [2] to determine the control functions $d_5(t)$, $d_6(t)$, $u_{10}(t)$, and $u_{11}(t)$ for realizing the SWAP gate transformation,

$$V_{SW} = \begin{bmatrix} 1 & & & \\ & 0 & 1 & \\ & 1 & 0 & \\ & & & 1 \end{bmatrix} \in \mathbb{C}^{2\times 2} \tag{11}$$

Since higher energy levels play an important role in a cross-resonance gate, we model each transmon by 4 energy levels, leading to a state vector with $N = 16$ elements. In the closed system setting, the time-evolution of the quantum system is unitary, yielding the transformation $\psi(t) = U(t, \boldsymbol{\alpha})\psi(0)$ for any initial quantum state $\psi(0)$. Here, $\boldsymbol{\alpha} \in \mathbb{C}^D$ is the vector of control parameters and $U(t, \boldsymbol{\alpha}) \in C^{N\times N}$ is the unitary solution matrix, which solves Schrödinger's equation

$$\dot{U}(t, \boldsymbol{\alpha}) = -i\widetilde{H}(t, \boldsymbol{\alpha})U(t, \boldsymbol{\alpha}) \quad 0 < t \leq T, \quad \text{with} \quad U(0) = I_N \tag{12}$$

where I_N is the $N \times N$ identity matrix and $\widetilde{H}(t, \boldsymbol{\alpha}) = \widetilde{H}_{sys} + \widetilde{H}_c(t, \boldsymbol{\alpha})$ denotes the Hamiltonian in the rotating frame.

The main target of the optimization is to find the vector of control parameters $\boldsymbol{\alpha}$ that minimize the difference between the target SWAP gate matrix V_{SW} and the final-time solution operator $U(T, \boldsymbol{\alpha})$, projected onto the two lowest energy levels of each transmon. The difference between the matrices is measured in terms of the trace infidelity,

$$\mathcal{J}_1(\boldsymbol{\alpha}) = 1 - \frac{1}{N^2}\,\text{Tr}\left(V_{SW}^\dagger \widetilde{U}(T, \boldsymbol{\alpha})\right), \quad \widetilde{U} = P^\dagger U P. \tag{13}$$

Here, $U(t, \boldsymbol{\alpha})$ solves Schrödinger's equation (12) and $P^\dagger$ extracts the first two energy levels of each transmon from the full state vector. The time-averaged population of the highest energy level in each transmon, $\mathcal{J}_2(\boldsymbol{\alpha})$, is added to discourage leakage to non-computational levels of the transmons. The optimizer minimizes the total objective function $\mathcal{J}_1 + \mathcal{J}_2$, subject to constraints on the amplitude of the control functions, see [4] for further details.

Both Juqbox and Quandary represent the control Hamiltonian in terms of their real and imaginary components,

$$\widetilde{H}_{ctrl}(t) = p_5(t)(b_5 + b_5^\dagger) + iq_5(t)(b_5 - b_5^\dagger) + p_6(t)(b_6 + b_6^\dagger) + iq_6(t)(b_6 - b_6^\dagger) \tag{14}$$

To identify the relation between the real-valued control functions (p_k, q_k) and the complex-valued functions d_5, d_6, u_{10} and u_{11} in (10), it is convenient to first introduce the complex-valued functions $\zeta_k(t)$,

$$\zeta_k(t) = \sum_{\ell=1}^{N_s} \alpha_{k,\ell}\hat{B}_\ell(t), \quad \alpha_{k,\ell} \in \mathbb{C}. \tag{15}$$

In Juqbox, $\hat{B}_\ell(t)$, $\ell = 1, 2, \ldots, N_s$, are quadratic B-spline wavelets, uniformly spaced in time. We define,

$$p_5(t) + iq_5(t) := e^{i\delta_5 t}\zeta_5(t) + e^{i\delta_6 t}\zeta_{10}(t) \tag{16}$$

$$p_6(t) + iq_6(t) := e^{i\delta_6 t}\zeta_6(t) + e^{i\delta_5 t}\zeta_{11}(t) \tag{17}$$

Because there are four control functions, the total number of control parameters becomes $D = 4N_s$. In this work, 30 B-spline coefficients were used to parameterize each control function. We remark that the number of control parameters can be chosen independently of (and usually much smaller than) the number of time steps for integrating Schrödinger's equation (In this study we used $N_T = 105,625$ time steps to integrate Schrödinger's equation to time $T = 668.4$ ns).

The functions $\zeta_k(t)$ allow the control Hamiltonian (14) to be written as

$$\widetilde{H}_{c,c}(t) = \left(e^{i\delta_5 t}\zeta_5(t) + e^{i\delta_6 t}\zeta_{10}(t)\right) b_5 + \left(e^{i\delta_6 t}\zeta_6(t) + e^{i\delta_5 t}\zeta_{11}(t)\right) b_6 + \text{H.c.} \tag{18}$$

The un-calibrated relation between the control functions in Qiskit and Juqbox is found by comparing (18) and(10),

$$d_5(t) = \frac{2}{\Omega_{d,5}}\zeta_5(t) \tag{19}$$

$$d_6(t) = \frac{2}{\Omega_{d,6}}\zeta_6(t) \tag{20}$$

$$u_{10}(t) = \frac{2}{\Omega_{d,5}}\zeta_{10}(t) \tag{21}$$

$$u_{11}(t) = \frac{2}{\Omega_{d,6}}\zeta_{11}(t) \tag{22}$$

3.1 Open system optimal control

To account for system-environment interactions, the numerical optimization with Juqbox can be used as a starting point for optimal control with the Quandary code [2]. Quandary describes open quantum systems using the density matrix $\rho \in \mathbb{C}^{N\times N}$. The time evolution of $\rho(t)$ is modelled by Lindblad's master equation

$$\dot{\rho}(t) = -\,i\,(H(t)\rho(t) - \rho(t)H(t)) + L\,(\rho(t)) \tag{23}$$

Both decay and dephasing processes are modeled using the Lindblad terms:

$$L(\rho) = \sum_{k=5}^{6}\sum_{l=1}^{2} \mathcal{L}_{lk}\rho\mathcal{L}_{lk}^\dagger - \frac{1}{2}\left(\mathcal{L}_{lk}^\dagger\mathcal{L}_{lk}\rho + \rho\mathcal{L}_{lk}^\dagger\mathcal{L}_{lk}\right) \tag{24}$$

where the collapse operators satisfy $\mathcal{L}_{1k} := \frac{1}{\sqrt{T_1^k}}a_k$ (decay) and $\mathcal{L}_{2k} := \frac{1}{\sqrt{T_2^k}}a_k^\dagger a_k$ (dephasing). Here, T_1^k and T_2^k correspond to the decay and dephasing times for system k.

Quandary solves Lindblad's master equation numerically applying the implicit midpoint time-stepping method, which is a symplectic, second-order time-integration scheme of Runge-Kutta type. In order to derive the discrete adjoint equations, techniques from Algorithmic Differentiation are applied to yield consistent and exact gradients of the objective function at costs that are independent of the number of control parameters. The optimization problem is the solved iteratively using gradient updates, preconditioned by L-BFGS updates to incorporate Hessian information. Constraints on the maximum amplitudes of the control parameters are incorporated using a projected line-search procedure that project the gradient onto the linear box-constraints of maximally allowed amplitudes.

4 Rabi pulse calibrations

Qiskit support two types of channels that accept custom pulses: drive and control channels. Pulses on the drive channels are specified in the following way:

$$D_i(t_j) = \mathrm{Re}\left\{\exp(i2\pi f j dt + \phi) d_j\right\}. \tag{25}$$

Here f is a frequency that can be chosen by a user. It defaults to the qubit transition frequency for drive channels and the frequency of the linked qubit for control channels. ϕ is a phase, the time step is set to $dt = 2/9$ ns for the Casablanca system, and d_j is the non-dimensional, complex-valued control amplitude at time $t_j = j\,dt$. For each channel, Qiskit specifies the maximum allowable amplitude signal, $\Omega_{d,i}$. If we absorb the phase into the dimensionless amplitude via $\tilde{d}_j = e^{i\phi} d_j$, then

$$\Omega_{d,i} D_i(t_j) = \Omega_{d,i} \left(\cos(2\pi f j dt)\mathrm{Re}\{\tilde{d}_j\} - \sin(2\pi f j dt)\mathrm{Im}\{\tilde{d}_j\}\right),$$

which gives the basic mapping between Qiskit and Juqbox:

$$p(t_j) = \frac{\mathrm{Re}\{\tilde{d}_j\}}{2\Omega_{d,i}}, \quad q(t_j) = \frac{\mathrm{Im}\{\tilde{d}_j\}}{2\Omega_{d,i}}.$$

To verify this mapping, we considered a Rabi pulse for a single qubit. With a single carrier-wave with zero frequency in the rotating frame and two energy levels, with constant $p = A_J, q = 0$, half a Rabi oscillation occurs in Juqbox simulations for the pulse duration

$$\tau_p = \frac{\pi}{|A_J|} \implies |A_J| = \frac{\pi}{\tau_p}.$$

Using Qiskit's amplitude convention, the corresponding relation is

$$A_Q = \mathrm{Re}\{\tilde{d}_j\} = \frac{2\pi}{\tau_p}\Omega_{d,i}.$$

The ratio between the amplitudes of the corresponding lab frame control signals in Qiskit and Juqbox becomes

$$\frac{|A_Q|}{2|A_J|} = \Omega_{d,i}.$$

This implies that the amplitudes should be proportional to each other. Results from a Rabi experiment on the Casablanca hardware are shown in Figure 1.

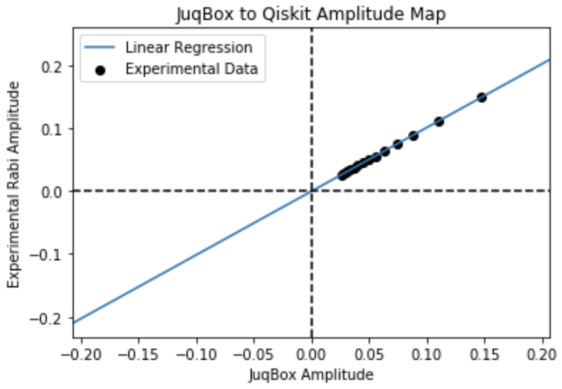


Figure 1: Results of the Rabi experiment on qubit 5 of the Casablanca hardware.

For the Casablanca hardware, the fitted line $y = 1.014x - 1.9 \cdot 10^{-4}$ has a slope that is close to $\Omega_{d,5} \approx 1.084$. This study confirms that the drive channel amplitude $\Omega_{d,5}$ is almost perfectly calibrated.

5 Gaussian square and DRAG pulses in Qiskit

We start by defining the zeroed Gaussian function, centered at time $T/2$:

$$g_z(t; A, T, \sigma) = \begin{cases} A\left(\exp\left(-\frac{(t-T/2)^2}{2\sigma^2}\right) - g_0(T,\sigma)\right), & t \in [0, T], \\ 0, & \text{otherwise.} \end{cases}$$

Here A is a complex-valued amplitude, T is the duration of the zeroed Gaussian, and σ is its standard deviation. After digging deep into the Qiskit source code, we found that the tails of the Gaussian are truncated by subtracting out the constant g_0, defined by

$$g_0(T, \sigma) = \exp\left(-\frac{(T/2 + dt)^2}{2\sigma^2}\right), \quad \text{where } dt = \tfrac{2}{9} \text{ ns.} \tag{26}$$

Based on the zeroed Gaussian pulse, we can now define the Derivative Removal by Adiabatic Gate (DRAG) pulse:

$$f(t; A, T, \sigma, \beta) = g_z(t; A, T, \sigma) + i\beta \underbrace{\left(-\frac{(t - T/2)}{\sigma^2}\right) g(t; A, T, \sigma)}_{g_z'(t)},$$

where β is a correction amplitude. Finally, the Gaussian square pulse is defined by its amplitude A, total duration T, and the duration of its constant part, w. Let the duration of the leading and trailing ramp be $r = (T - w)/2 > 0$. Then,

$$s(t; A, T, \sigma, w) = \begin{cases} \dfrac{g_z(t; A, 2r, \sigma)}{1 - g_0(2r, \sigma)}, & 0 \le t \le r, \\ A, & r \le t \le r + w, \\ \dfrac{g_z(T - t; A, 2r, \sigma)}{1 - g_0(2r, \sigma)}, & r + w \le t \le T. \end{cases}$$

6 Converting Qiskit pulses to B-splines with carrier waves

For many basic gates Qiskit provides parametric representations of the pulse schedule required to realize the chosen gate. These provided pulses take the form (25). For simplicity, suppose we have the signal D_0 and wish to represent it in Juqbox/Quandary via B-splines with carrier waves. Since each signal in Qiskit is associated with a single frequency we have

$$\Omega_{d,0} D_0(t_j) = \Omega_{d,0} \left(\cos(2\pi f j dt) \mathrm{Re}\{\tilde{d}_j\} - \sin(2\pi f j dt) \mathrm{Im}\{\tilde{d}_j\} \right) = 2p(t_j)\cos(\omega t_j) - 2q(t_j)\sin(\omega t_j).$$

The approximation of the real and imaginary parts of d_j can be done independently, so that in the following we focus on approximating the real part. This becomes a classical interpolation problem

$$\frac{\Omega_{d,0}}{2} \mathrm{Re}\{d_k\} = p(t_k) = \sum_{j=1}^{D_1} \alpha_j B_j(t_k), \quad \forall k = 1, 2, \ldots, N,$$

where N is the number of samples d_k. This gives us the Vandermonde system

$$\underbrace{\begin{pmatrix} B_1(t_1) & B_2(t_1) & B_3(t_1) & \cdots & B_{D_1}(t_1) \\ B_1(t_2) & B_2(t_2) & B_3(t_2) & \cdots & B_{D_1}(t_2) \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ B_1(t_N) & B_2(t_N) & B_3(t_N) & \cdots & B_{D_1}(t_N) \end{pmatrix}}_{\mathbb{R}^{N \times D_1}} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{D_1} \end{pmatrix} = \frac{\Omega_{d,0}}{2} \begin{pmatrix} \mathrm{Re}\{d_1\} \\ \mathrm{Re}\{d_2\} \\ \vdots \\ \mathrm{Re}\{d_N\} \end{pmatrix},$$

where D_1 is the number of B-splines used in the approximation. Given that the time points t_k are distinct and $D_1 = N$ the above system is uniquely solvable. We also note that at a given time t_k only three B-splines are non-zero so that for a large number of splines, D_1, the above system is sparse. Moreover, if we choose the interpolation points t_k to be the centers of each B-spline and pick $D_1 = N$ then the above is the tridiagonal system

$$\begin{pmatrix} 3/4 & 1/8 & & & \\ 1/8 & 3/4 & 1/8 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & 1/8 \\ & & & 1/8 & 3/4 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{D_1-1} \\ \alpha_{D_1} \end{pmatrix} = \frac{\Omega_{d,0}}{2} \begin{pmatrix} \mathrm{Re}\{d_1\} \\ \mathrm{Re}\{d_2\} \\ \vdots \\ \mathrm{Re}\{d_{D_1-1}\} \\ \mathrm{Re}\{d_{D_1}\} \end{pmatrix},$$

which can be solved in $\mathcal{O}(D_1)$ time.

7 Reverse model calibration using X- and Cx-gates

The Casablanca system uses DRAG pulses to implement X-gates and a combination of Gaussian square and DRAG pulses to implement Cx gates. As the pulse coefficients are updated during each calibration of the system hardware, they can be used to calibrate the computational model for the Casablanca hardware. Given the high fidelity of the corresponding gates, the pulses can be used to engineer a mapping between the computational control functions we optimize with Juqbox and the physical control functions that must be applied to the hardware.

7.1 X-gates

The X-gate for qubit 5 is defined by the single DRAG pulse

$$d_5(t) = f(t; A_1, T, \sigma, \beta_1),$$

for $0 \leq t \leq 160 \cdot dt$ where $dt = 2/9$ ns. The X-gate for qubit 6 is also defined by the single DRAG pulse

$$d_6(t) = f(t; A_2, T, \sigma, \beta_2),$$

for $0 \leq t \leq 160 \cdot dt$. The parameters defining each DRAG pulse can be accessed using Qiskit by querying the backend for Casablanca. The coefficients change slightly after each system calibration, but the duration of the DRAG pulses appears to be fixed at $160 \cdot dt$. At some point during the spring of 2021, the coefficients were

$$\begin{aligned} A_1 &= 0.17545065110530234, \\ A_2 &= 0.20674287767710134, \\ T &= 160, \\ \sigma &= 40, \\ \beta_1 &= 0.47609887200679674, \\ \beta_2 &= 1.9314472856919194. \end{aligned}$$

The pulse schedules for the above X-gates are shown in Figure 2.

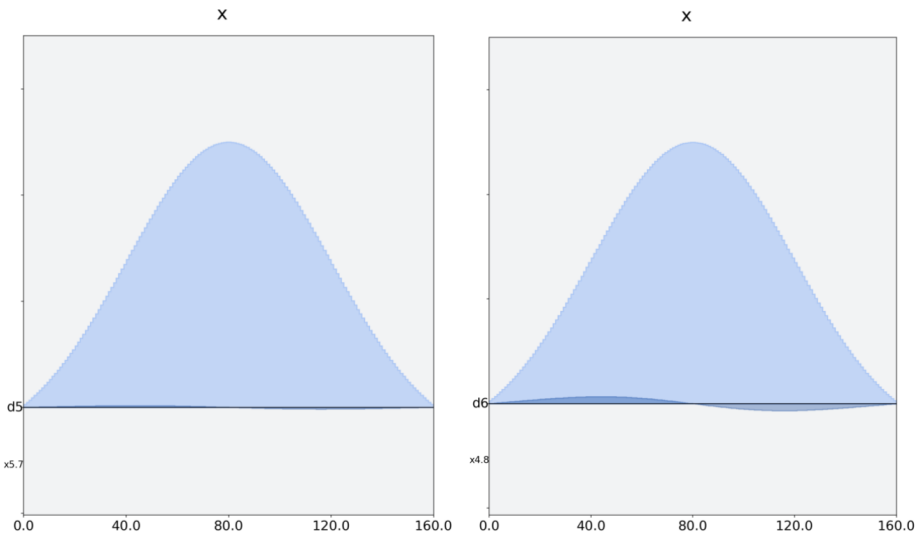


Figure 2: The pulse schedule for an X gate on Casablanca for qubit 5 (left), and qubit 6 (right).

Only the $d_5(t)$ function is active during the X5-gate and only the $d_6(t)$ function is active during the X6-gate, which allows the pulses to be calibrated independently. We want to improve the calculated average gate fidelity by modifying the control Hamiltonian (10) according to

$$d_5(t) \to \xi_5 d_5(t), \tag{27}$$
$$d_6(t) \to \xi_6 d_6(t), \tag{28}$$

where ξ_5 and ξ_6 are calibration factors. The gate fidelity is measured by

$$F_{avg} = \frac{1}{N}\sum_{j=1}^{N} \mathrm{Tr}|V^\dagger U(T,\boldsymbol{\alpha})|. \tag{29}$$

Here V is the target unitary and U is the unitary solution matrix for Schrödinger's equation at final time T. The highest X-gate fidelities were obtained for the coefficients

$$\xi_5 = 0.9927, \quad \xi_6 = 0.9909, \tag{30}$$

indicating that the drive channels on the Casablanca system are very well calibrated. We remark that $\xi_5\Omega_{d,5} \approx 1.08$ is close to the slope 1.014 that was found in the Rabi calibration experiment in Section 4.

7.2 Calibrating a cross-talk model using the Cx gates

The Casablanca backend holds calibrated pulse sequences for two CX-gates that involve qubits 5 and 6: CX-56 (5 controls 6) and CX-65 (6 controls 5). These gates are implemented with a combination of DRAG and Gaussian square pulses and use the control functions d_5, d_6 and u_{11} (but not u_{10}), see Figure 3. Because the control function u_{11} acts on qubit 6 but uses qubit 5's frequency, the corresponding signal may be subject to cross-talk. Following [3], we can account for this effect through the simple model

$$u_{11}(t)b_6 \to \xi_{11}u_{11}(t)b_6 + A_c e^{i\phi_c}u_{11}(t)b_5, \tag{31}$$

where ξ_{11} and A_c are cross-talk coefficients and ϕ_c is a phase shift that compensates for the physical distance between qubits 5 and 6 on the chip. After sweeping through parameter space, we found that

$$\xi_{11} = 2.02, \quad A_c = 0.0583, \quad \phi_c = 1.2189, \tag{32}$$

result in gate fidelities of 0.994 for both the CX-56 gate and the CX-65 gate.

To complete our modelling we also need to handle potential cross-talk from control function u_{10}. Unfortunately, we have been unable to locate any calibrated gates that use this control channel on the Casablanca system. In lieu of calibrated pulses, we use a symmetry argument to motivate the same cross-talk model for u_{10} as for u_{11}, i.e.,

$$u_{10}(t)b_5 \to \xi_{10}u_{10}(t)b_5 + A_c e^{i\phi_c}u_{10}(t)b_6, \quad \xi_{10} = \xi_{11}. \tag{33}$$

As a result of the above calibrations, the modified Hamiltonian model becomes $\widetilde{H}_{comp}(t) = \widetilde{H}_{sys} + \widetilde{H}_{c,c}(t)$, where

$$\begin{aligned}\widetilde{H}_{c,c}(t) = &\frac{\Omega_{d,5}}{2}\left(e^{i\delta_5 t}\xi_5 d_5(t)b_5 + e^{i\delta_6 t}u_{10}(t)(\xi_{10}b_5 + A_c e^{i\phi_c}b_6)\right)\\ &+ \frac{\Omega_{d,6}}{2}\left(e^{i\delta_6 t}\xi_6 d_6(t)b_6 + e^{i\delta_5 t}u_{11}(t)(\xi_{11}b_6 + A_c e^{i\phi_c}b_5)\right) + \text{H.c.}\end{aligned} \tag{34}$$

The Juqbox code represents the control Hamiltonian in terms of its real and imaginary components, as described above. By identifying the coefficients between (18) and (34), we arrive at the

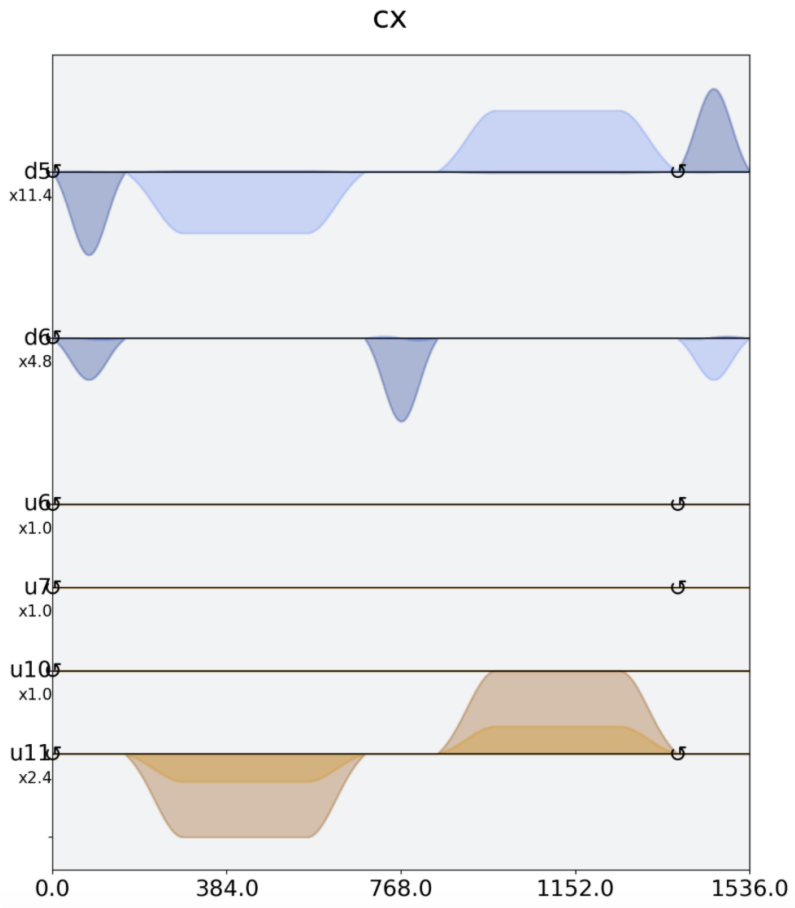


Figure 3: The pulse schedule for a CNOT gate on Casablanca where qubit 5 is the control qubit and qubit 6 is the target.

calibrated conversion

$$\zeta_5(t) = \frac{\Omega_{d,5}}{2}\xi_5 d_5(t) + \frac{\Omega_{d,6}}{2}A_c e^{i\phi_c}u_{11}(t) \tag{35}$$

$$\zeta_6(t) = \frac{\Omega_{d,6}}{2}\xi_6 d_6(t) + \frac{\Omega_{d,5}}{2}A_c e^{i\phi_c}u_{10}(t) \tag{36}$$

$$\zeta_{10}(t) = \frac{\Omega_{d,5}}{2}\xi_{10} u_{10}(t) \tag{37}$$

$$\zeta_{11}(t) = \frac{\Omega_{d,6}}{2}\xi_{11} u_{11}(t) \tag{38}$$

Thus, control functions $(\zeta_5, \zeta_6, \zeta_{10}, \zeta_{11})$ that are optimized with Juqbox should be converted to

Qiskit according to

$$d_5(t) = \frac{2}{\Omega_{d,5}\xi_5}\left(\zeta_5(t) - A_c e^{i\phi_c}\frac{1}{\xi_{11}}\zeta_{11}(t)\right) \tag{39}$$

$$d_6(t) = \frac{2}{\Omega_{d,6}\xi_6}\left(\zeta_6(t) - A_c e^{i\phi_c}\frac{1}{\xi_{10}}\zeta_{10}(t)\right) \tag{40}$$

$$u_{10}(t) = \frac{2}{\Omega_{d,5}\xi_{10}}\zeta_{10}(t) \tag{41}$$

$$u_{11}(t) = \frac{2}{\Omega_{d,6}\xi_{11}}\zeta_{11}(t) \tag{42}$$

The control pulses used by Qiskit are defined by inserting the above functions into (5)-(6).

8 Implementation of custom pulses in Qiskit

In Juqbox/Quandary, each control signal is represented by a continuous approximation of B-splines with carrier waves. With the above mapping we simply evaluate the control signals at equispaced time points, $t_k = k\,dt$ where $dt = 2/9$ ns, and then non-dimensionalize. Once we have a set of dimensionless amplitude samples, we need to make Qiskit aware of our custom pulse. Our chosen approach is to simply pass a non-basis element Clifford gate (such as the SWAP gate) to `randomized_benchmarking_seq` as follows:

```
# Use standard swap gate for interleaved circuit
circ = QuantumCircuit(2)
circ.swap(0,1)
interleaved_elem = [circ]

# generate the RB circuit parameters
length_vector = np.arange(1,200,20)
nseeds = 5

rb_pattern = [[5,6]]

# Generate the RB circuits
_,_circs = randomized_benchmarking_seq(length_vector=length_vector,
                                       nseeds=nseeds,
                                       rb_pattern=rb_pattern,
                                       interleaved_elem=interleaved_elem)
```

With the interleaved circuits now built, we can manually go through and add our custom pulse to each SWAP gate as follows:

```
for circuits in circs:
    for circuit in circuits:
        circuit.add_calibration("swap", qubits=[5, 6], schedule=sched)
```

where `sched` is a `Schedule` object containing our custom pulse. This approach allows us to avoid issues with `randomized_benchmarking_seq` decomposing a custom gate while providing a custom pulse that the transpiler respects when executing the sequence on the actual device. When downloading the associated JSON files with each RB job on IBM Quantum using this approach, it

can be seen that each custom pulse is assigned to a 64 bit hexadecimal string indicating the signal is being used on the actual device.

A final practical matter with this approach is that directly adding the full custom waveform as a single instruction when building a schedule can result in the error:
`Waveform memory exceeds the maximum amount of memory currently available # [8018]`.
To avoid this error, we have built the full schedule by splitting each custom waveform in smaller chunks. We are currently using a chunk size of at most 160 complex-valued amplitudes.

9 Randomized benchmarking results for two models

To reduce influence of state preparation and measurement (SPAM) errors, the fidelity of the SWAP gate is estimated using randomized benchmarking, as provided by the `randomized_benchmarking_seq()` function in Qiskit. Here, two sets of randomized circuits are constructed. The first one provided a reference and is composed of a standard gate set. In the second one, the standard gate set is augmented by an interleaved circuit, which is this case corresponds to a custom SWAP gate. Before transpiling the interleaved circuits, our custom pulse schedule was inserted by explicitly calling `add_calibration()`, as described above. An example of the pulse schedule is shown in Figure 4.

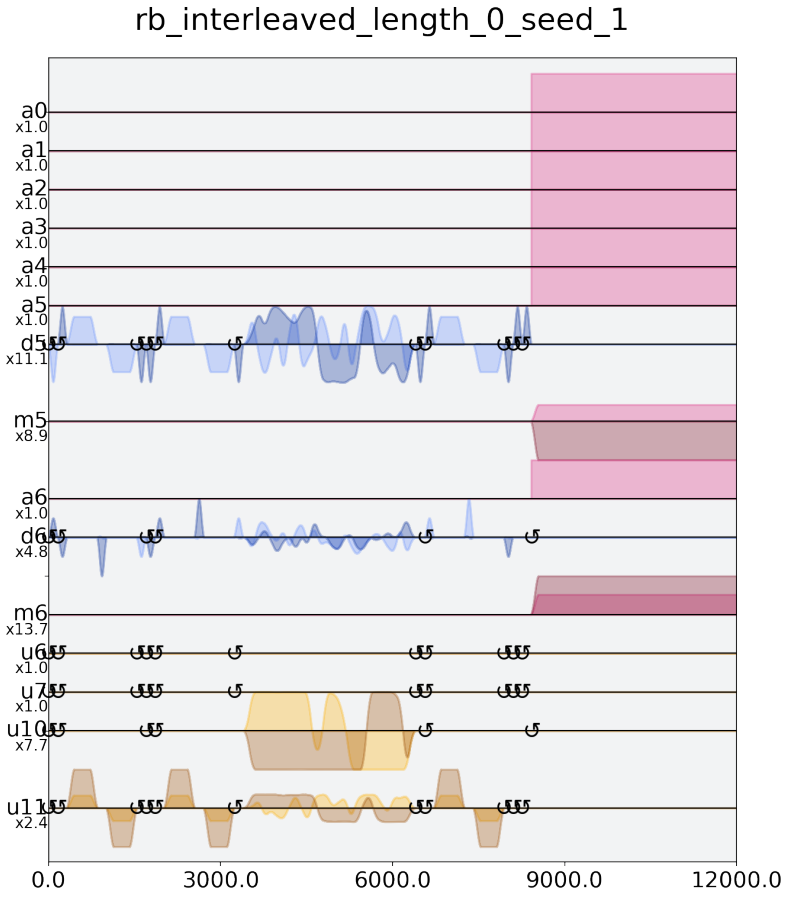


Figure 4: The pulse schedule for the first interleaved RB schedule.

The results of the non-interleaved and interleaved randomized benchmarking are processed by the Qiskit function `InterleavedRBFitter()`. It fits the data for each case to an exponential function and estimates the SWAP gate error as the difference in exponential decay between the fitted functions. With 5 randomized samples per case, 1000 shots per circuit, and considering circuit lengths in the range of 1-181, the results are shown in Figure 5. The estimated SWAP gate error is 2.36%.

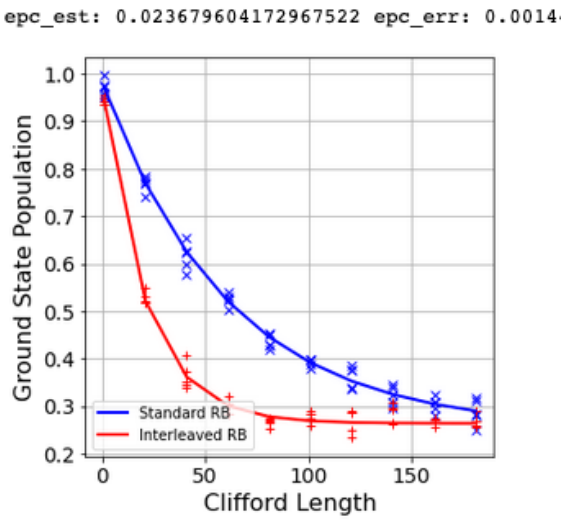


Figure 5: The observed ground state population as function of the length of the Clifford circuit. Here, the non-interleaved circuits are shown in blue and interleaved ones in red. The estimated error per Clifford (EPC) is 2.36%.

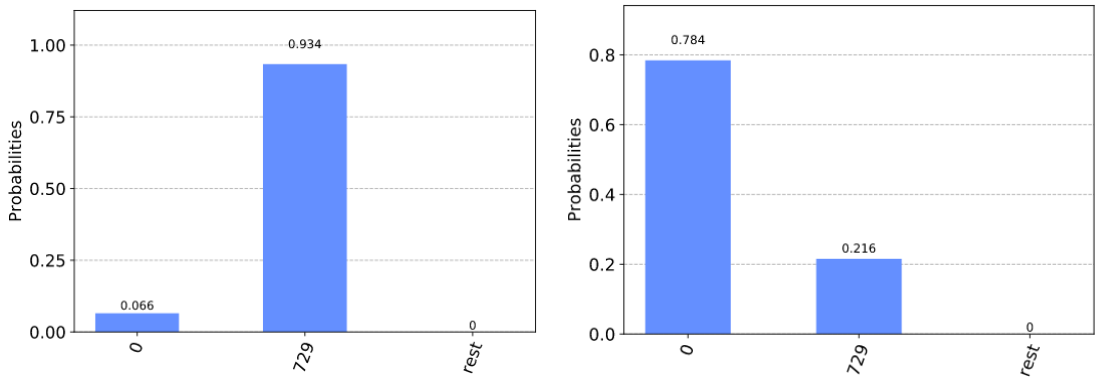
A Challenges encountered

Though the SWAP gate challenge provided an exciting competition and experience, we encountered several challenges during its course. Some problems were due to our unfamiliarity with Qiskit, but we also saw peculiar behavior of some of the functions in Qiskit, which hampered our progress. In this section, we summarize some of these challenges to help IBM improve their tools and documentation in the future.

A.1 A long job queue

The major challenge we have encountered was the extensive wait time between submitting a job on the Casablanca system and receiving the results. An average time of one cycle of submitting job to obtaining a result was about a week. Because our strategy was iterative (in a reverse engineering method), we could not proceed to the next step until we get results from IBM Casablanca system. Therefore, our natural next step was to use Casablanca pulse simulator, which does not require any queue.

A.2 Casablanca pulse simulator



(a) one X-gate applied on qubit 6

(b) two X-gates applied on qubit 6

Figure 6: Histogram counts over 1024 shots with the default X-gate from qiskit on Casablanca pulse simulator.

We were able to construct the Casablanca pulse simulator that is supposed to mimic the behavior of the actual Casablanca system, using the QasmSimulator.[1]. In order to verify the simulator, a simple experiment was conducted. First we created a channel with Qubits 5 and 6. Then we created a schedule with the default X-gate pulse on Qubit 6 and measurement on Qubit 5 and 6 with ground states, i.e., $|00\rangle$. Then we counted the measurements over 1024 simulations. Because there is noise and a simulator is supposed to mimic a real machine, the outcome of the measurements would not be 100 % $|10\rangle$. However, we expected to see a high probability on $|10\rangle$, which was labeled as 729 on the x-axis after the action of the X-gate.[2] Although we did see the high probability on $|10\rangle$, the level of the probability was not satisfactory, i.e., 0.934 in Figure 6(a). The fidelity further decreased as we added one more X-gate. For example, see Figure 6(b) where the number of counts for the ground state is only 0.784, indicating a low fidelity. The results were doubtful, so we moved on to the randomized benchmarking with the QasmSimulator to check the fidelity.

A.3 Custom Gate Class

Our first approach towards implementing a custom pulse on Casablanca was to create our own gate class for the custom SWAP gate. Below is the simple class which resembles Qiskit's `SwapGate` class:

```
class CustomSwapGate(Gate):
    def __init__(self, gate_name="my_gate", label=None):
        """Create new SWAP gate."""
        super().__init__(gate_name, 2, [], label=label)

    def _define(self):
```

[1] e.g., casablanca_simulator = QasmSimulator

[2] A side story: we were confused with the x-axis labels in the histogram. Because the system was supposed to be qubit, we only considered two states: one ground state and one excited state, and expected the label to follow the binary number system. However, it turns out that the label was based on ternary number system. Therefore, the following convention must be used: $0 = |00\rangle$, $243 = |01\rangle$, $729 = |10\rangle$, and $972 = |11\rangle$.

```
        """
        gate swap a,b { cx a,b; cx b,a; cx a,b; }
        """
        # pylint: disable=cyclic-import
        from qiskit.circuit.quantumcircuit import QuantumCircuit
        q = QuantumRegister(2, 'q')
        qc = QuantumCircuit(q, name=self.name)
        rules = [
            (CXGate(), [q[0], q[1]], []),
            (CXGate(), [q[1], q[0]], []),
            (CXGate(), [q[0], q[1]], [])
        ]
        for instr, qargs, cargs in rules:
            qc._append(instr, qargs, cargs)

        self.definition = qc

    def inverse(self):
        """Return inverse Swap gate (itself)."""
        return CustomSwapGate()  # self-inverse


    def to_matrix(self):
        """Return a numpy.array for the SWAP gate."""
        return numpy.array([[1, 0, 0, 0],
                            [0, 0, 1, 0],
                            [0, 1, 0, 0],
                            [0, 0, 0, 1]], dtype=complex)
```

We then may build a quantum circuit with a single `CustomSwapGate` instruction to pass to `randomized_benchmarking_seq` to create randomized benchmarking sequences as follows:

```
circ = QuantumCircuit(2)
circ.append(CustomSwapGate('csp'),[0,1])
interleaved_elem = [circ]

# generate the RB circuit parameters
length_vector = np.arange(1,200,20)
nseeds = 5

rb_pattern = [[5,6]]

# Generate the RB circuits
_,_,circs = randomized_benchmarking_seq(length_vector=length_vector,
                                        nseeds=nseeds,
                                        rb_pattern=rb_pattern,
                                        interleaved_elem=interleaved_elem)
```

Unfortunately, `randomized_benchmarking_seq()` decomposed our custom gate into three Cx gates via the provided `_define` decomposition. While this is useful to create inverse schedules necessary for randomized benchmarking, our custom gate was no longer in the circuit and the

custom sequence was then never used. Thankfully, the merging of PR #548 to into version 0.6 of Ignis added a new boolean argument, `keep_original_interleaved_elem`, which disallowed our custom gate to be further decomposed. With this update it became possible to have the transpiler attach our custom pulses to the circuits by adding our pulse to the InstructionScheduleMap. Below is the updated code snippet using this approach:

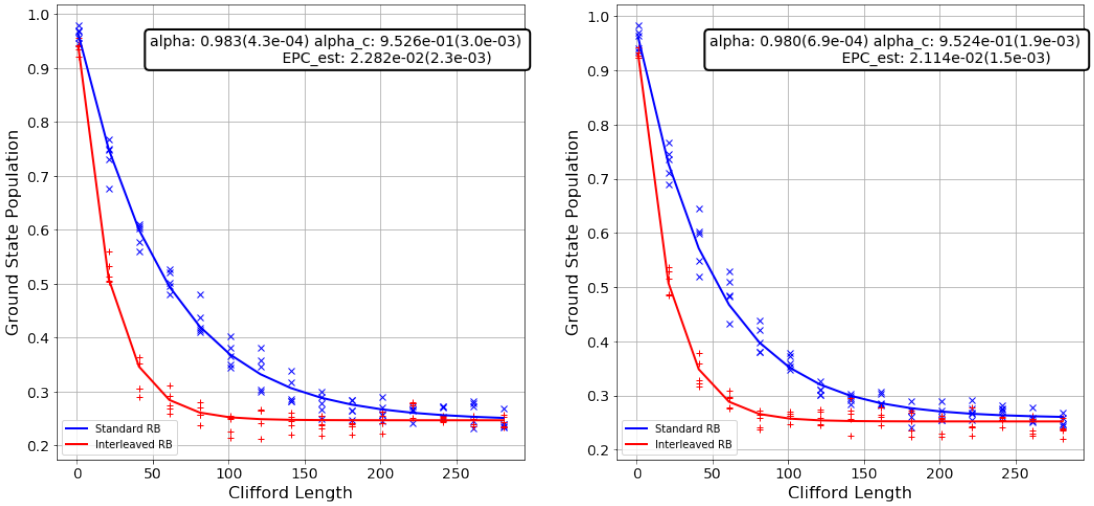
```
circ = QuantumCircuit(2)
circ.append(CustomSwapGate('csp'),[0,1])
interleaved_elem = [circ]

# generate the RB circuit parameters
length_vector = np.arange(1,200,20)
nseeds = 5

rb_pattern = [[5,6]]

# Generate the RB circuits
_,_,circs = randomized_benchmarking_seq(length_vector=length_vector,
                                        nseeds=nseeds,
                                        rb_pattern=rb_pattern,
                                        keep_original_interleaved_elem=True,
                                        interleaved_elem=interleaved_elem)
```

However, we note that the results from these circuits were odd and seemed to be the same regardless of the provided input signal as shown below in Figure 7.



(a) Optimized SWAP gate pulse sequence results

(b) Zero pulse sequence results

Figure 7: Randomized benchmarking results of custom pulses via the custom gate class approach for qubits 5 and 6 on Casablanca. Note that the EPC is about the same for both cases, though one uses an optimized pulse sequence and the other a pulse sequence of only zeros (free Hamiltonian evolution) which should do poorly in achieving a true SWAP gate.

Upon further inspection, the downloadable JSON files through the IBM Quantum site do not

contain the custom pulse definitions and defaulted back to the standard SWAP gate decomposition using 3 CX gates. The consistent performance regardless of the custom pulse sequences shown in Figure 7 are then likely due to the backend replacing custom pulse definitions with the usual SWAP gate used on Casablanca. We have verified that transpiling and drawing the pulse sequence for interleaved circuits generated in this manner contain our custom pulses, but appear to be ignored when they are submitted to the Casablanca device itself.

Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-TR-xxx). We gratefully acknowledge financial support from Lawrence Livermore National Laboratory through the Laboratory Directed Research and Development (LDRD) program, grant # 20-ERD-028.

Disclaimer

References

[1] Fortino Garcia and N. Anders Petersson. Juqbox. Github, 2021.

[2] Stefanie Günther and N. Anders Petersson. Quandary: Optimal control for open quantum systems. Github, 2021.

[3] E. Magesan and J. M. Gambetta. Effective Hamiltonian models of the cross-resonance gate. *Phys. Rev. A*, 101, 2020.

[4] N Anders Petersson, Fortino M Garcia, Austin E Copeland, Ylva L Rydin, and Jonathan L DuBois. Discrete adjoints for accurate numerical optimization with application to quantum control. *arXiv preprint arXiv:2001.01013*, 2020.