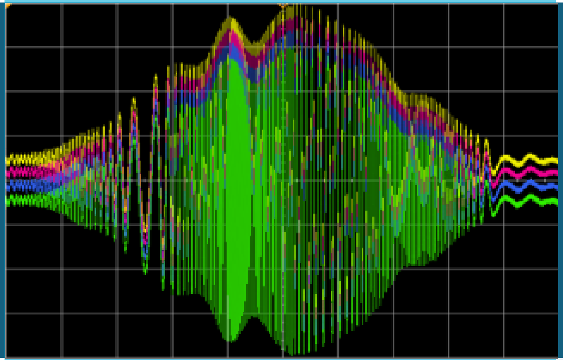
SAND2021-13170C

# Scalable coherent control hardware for trapped-ion systems: a tailored approach



PRESENTED BY

Daniel Lobser

## Hardware Implementation

QSCOUT

### General Requirements

- 2 tones per channel
  - Up to 350 MHz
- Absolute phase control
  - Re-synchronize phases without bookkeeping
- Fast & continuous parameter modulation
  - Spline modulation on all parameters simultaneously
- Synchronous control across channels
  - Full parallelism
- Long sequences of gates
- Scalable

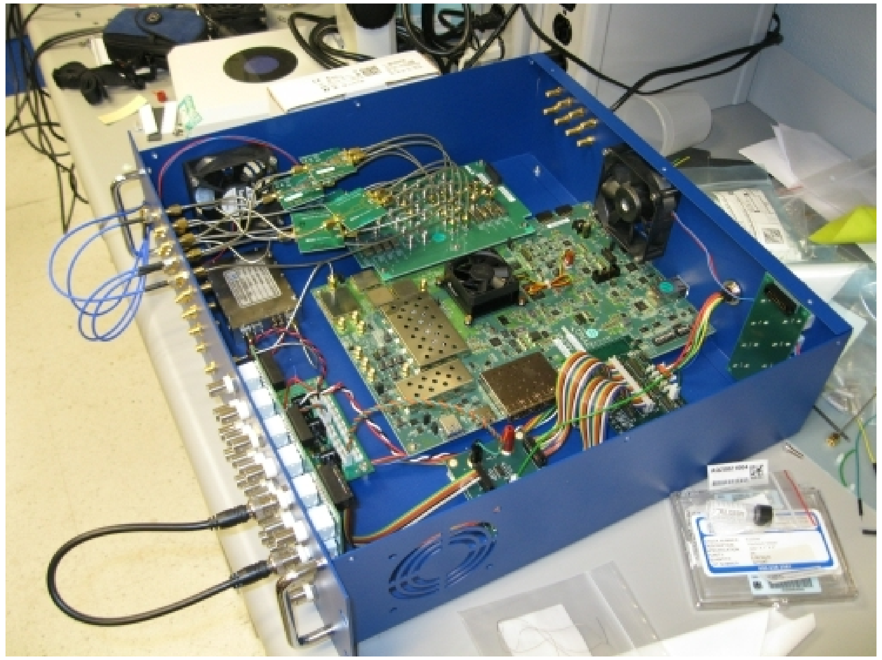
### Error Handling Requirements

- Fast repetition rate lock reconfigurability
- Cross talk cancellation

### Our Approach

- Custom firmware design using a Xilinx RFSoC
- Currently using ZCU111 evaluation boards
- 8 DACs per board, 6.5 GSPS

A Packaged "Octet"



## Gate Implementation at the Experiment Level

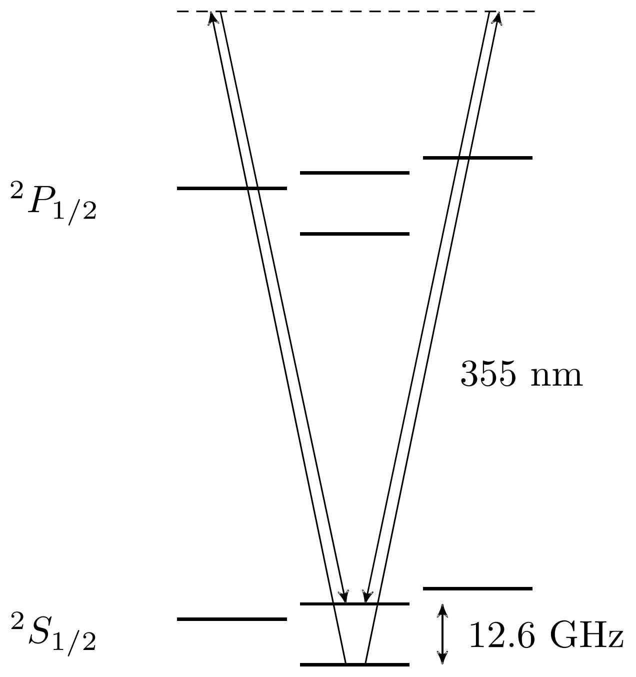
$^{171}$Yb$^{+}$ qubit, clock state 12.6 GHz

Individual addressing requires lasers

Optical frequency comb to bridges 12.6 GHz via Raman transitions

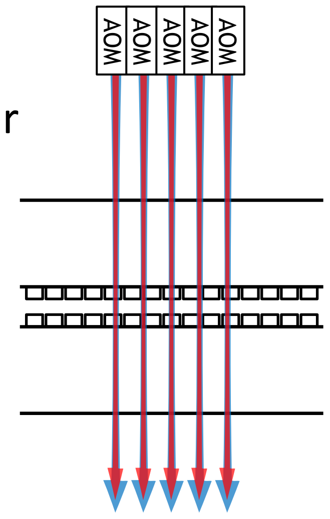
Frequency, phase, and amplitude control using RF signals applied to acousto-optic modulators (AOMs)

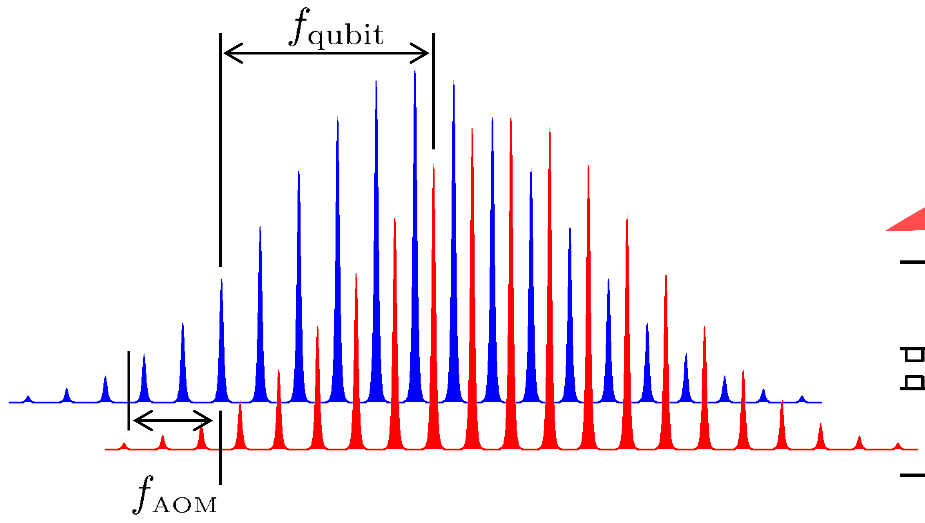
Two configurations: Co- and Counter-propagating

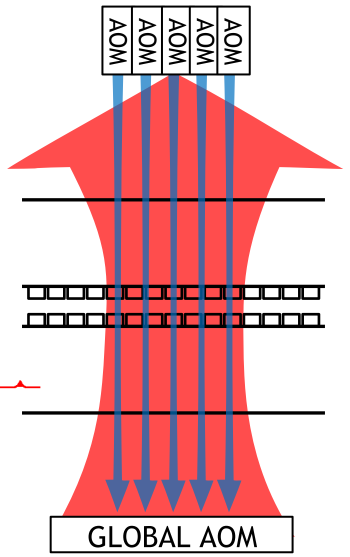


Co-propagating

- Immune to Doppler shifts
- Not affected by timing errors and pulse overlap





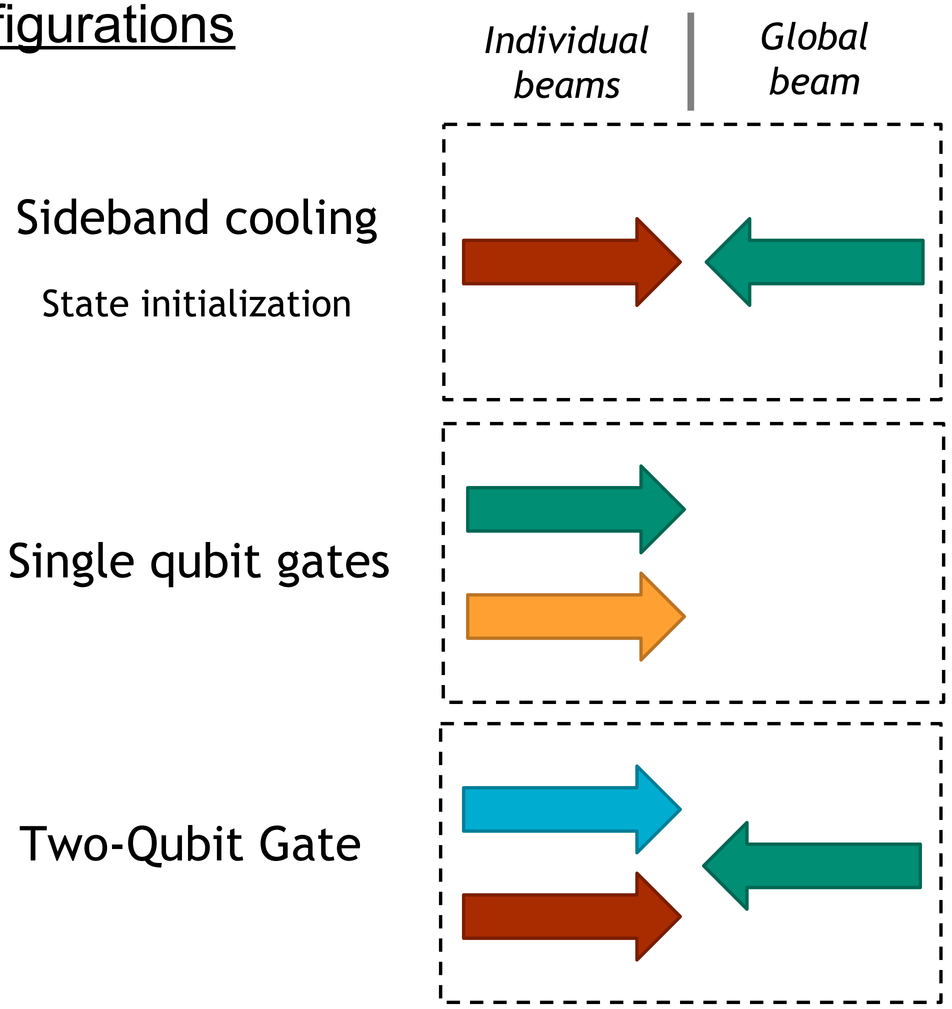


Counter-propagating

- Supports motional-state addressing and ground state cooling
- Affected by Doppler shifts
- Necessary for two-qubit gates

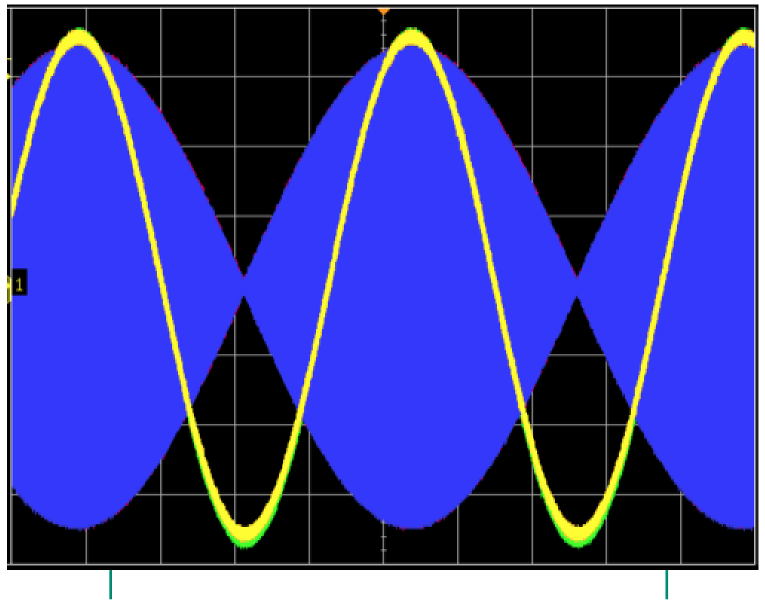
## Challenges: RF Reproducibility and Agility

Three basic configurations



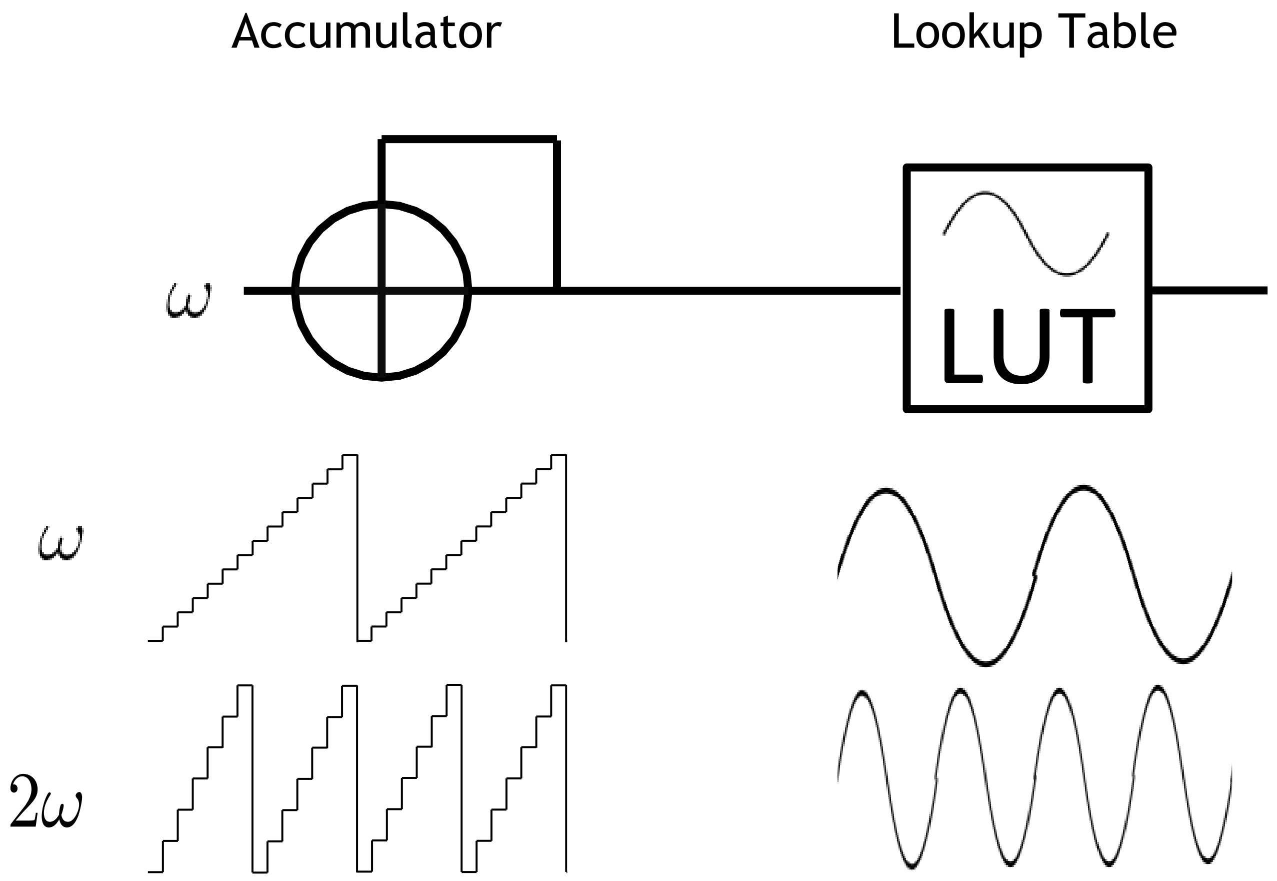
*Absolute phase control is imperative!*

- Each configuration requires different frequencies
- Phase of beat note produced by red and blue sideband tones determines global phase of Mølmer-Sørensen gate



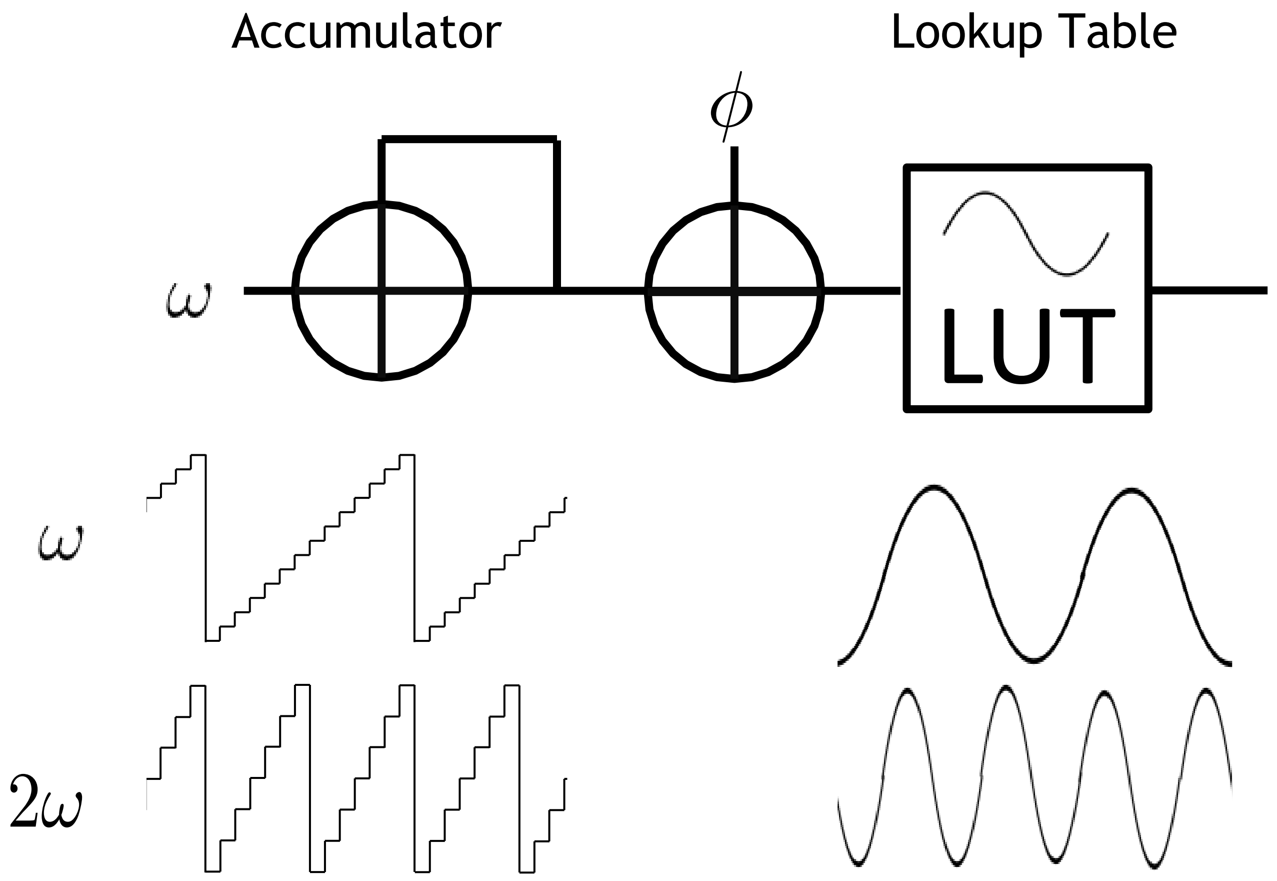
## Simplified Model of a Direct Digital Synthesizer (DDS)

Extremely simplistic model of a DDS requires an accumulator, which advances the phase of some waveform based on a frequency input, and something to convert the data into a sinusoidal amplitude such as a lookup table (LUT)



## Simplified Model of a Direct Digital Synthesizer (DDS)

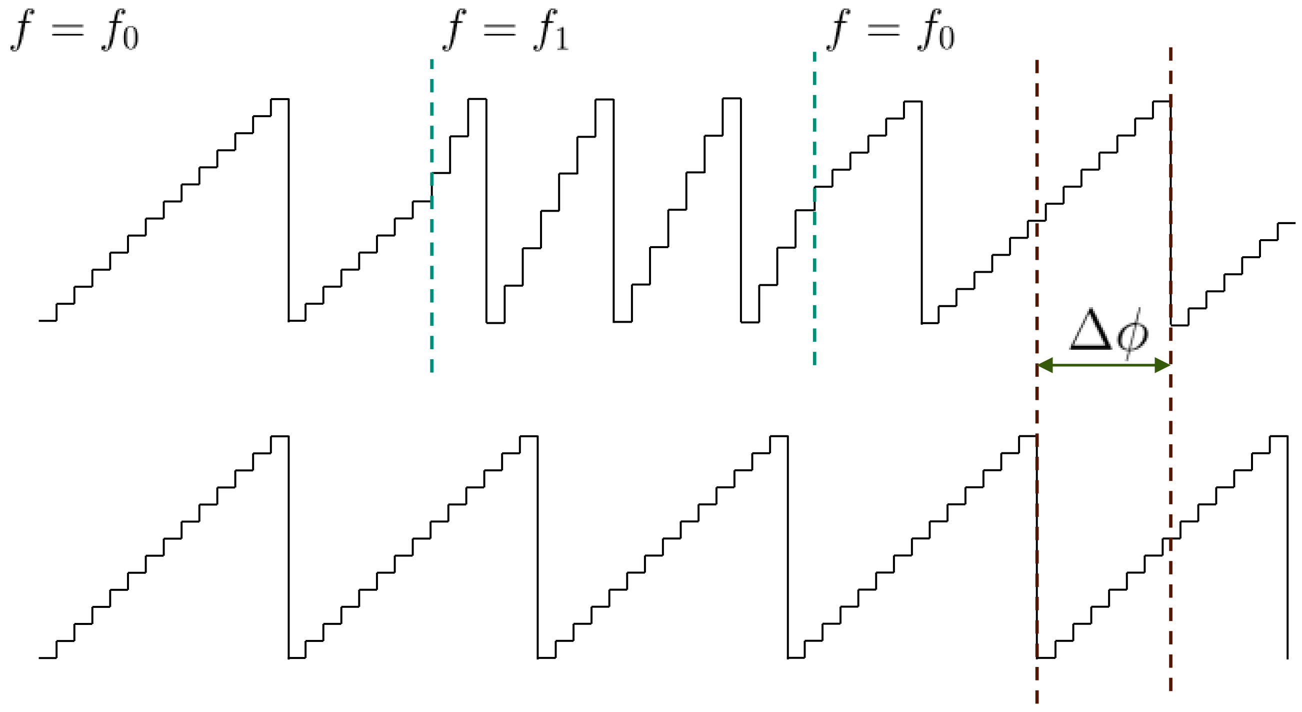
Extremely simplistic model of a DDS requires an accumulator, which advances the phase of some waveform based on a frequency input, and something to convert the data into a sinusoidal amplitude such as a lookup table (LUT)

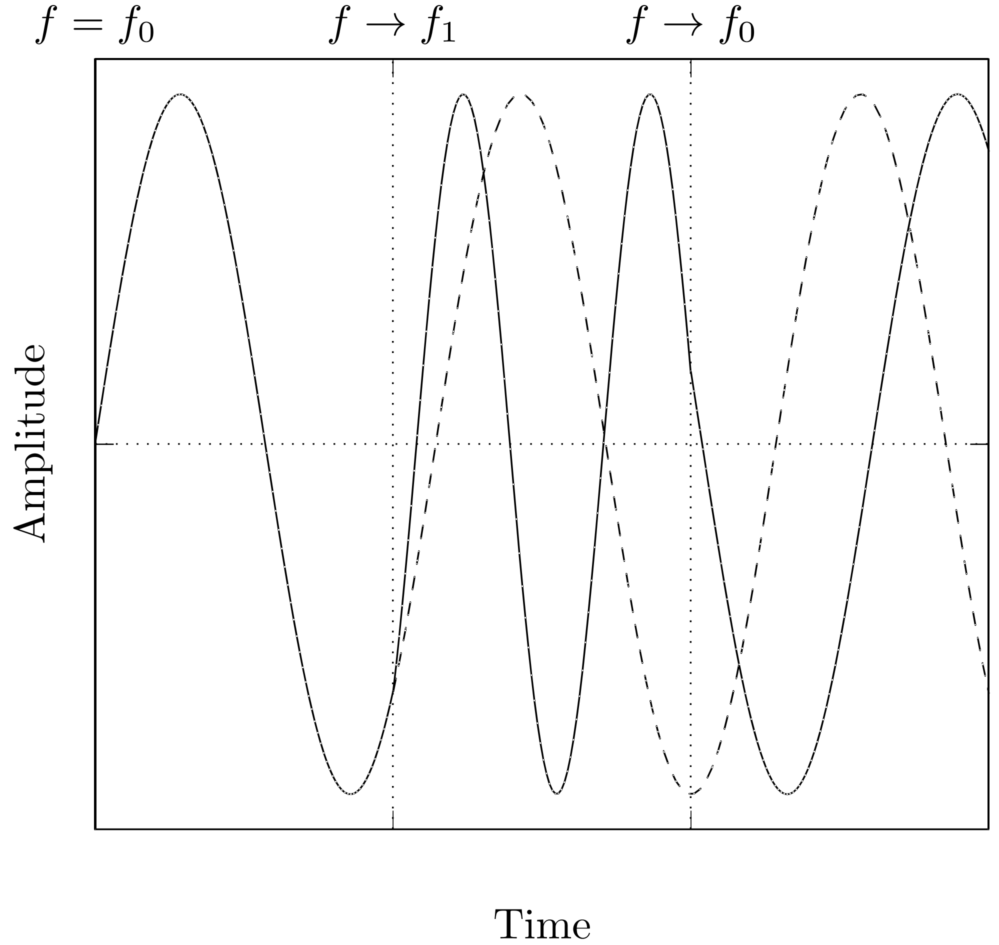


## Changing Frequency

Changes in the frequency word lead to continuous changes in the output, leading to undesirable phase differences when trying to maintain coherent gate operations.

One option is to use free-running DDSs and switching between them, but that typically wastes design resources or involves complex external switch networks that suffer from frequency-dependent phase shifts



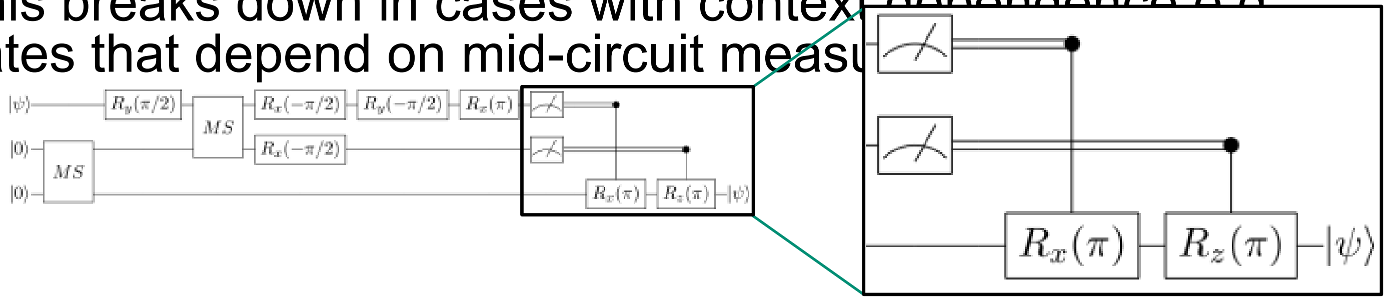


## Recycling Accumulators

DDS frequencies regularly need to be changed for running different types of gates → Accumulators need to be reset

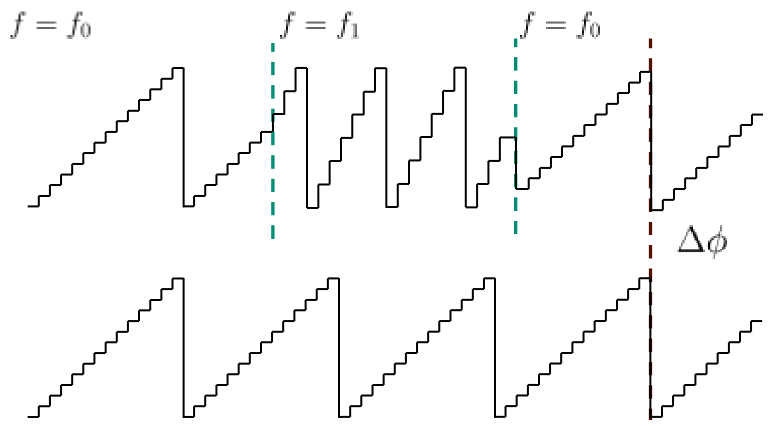
Manual phase bookkeeping requires pre-calculating phase based on when a particular gate is to be run during a sequence

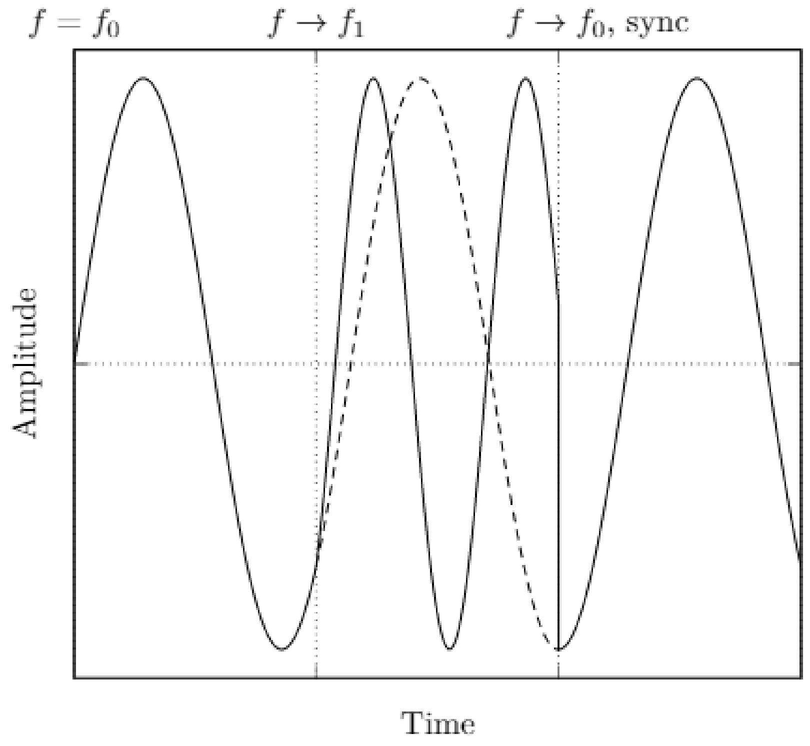
- This breaks down in cases with context dependence e.g. gates that depend on mid-circuit measu



  …or it leads to lots of effort making sure all gates have uniform time etc.

- This approach can be disrupted by small timing mismatch such as catching a trigger associated with a measurement result
- Extra time taken to calculate all of this information can subtly increase the time taken to compile all the pulse information





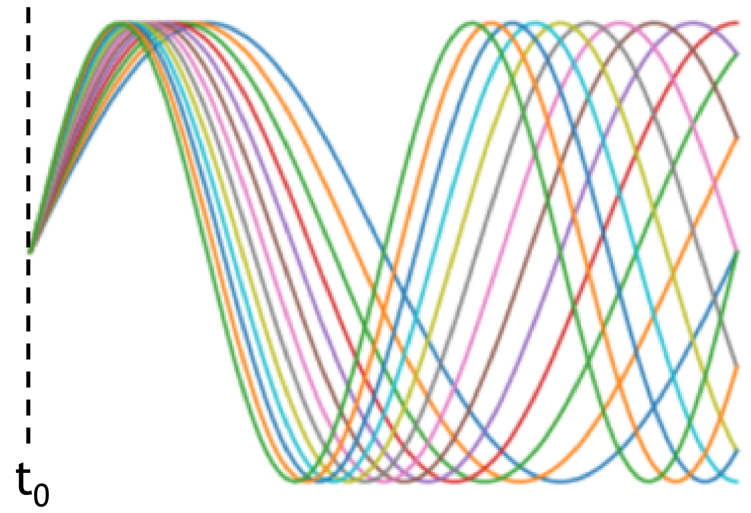
## Automated Global Phase Synchronization

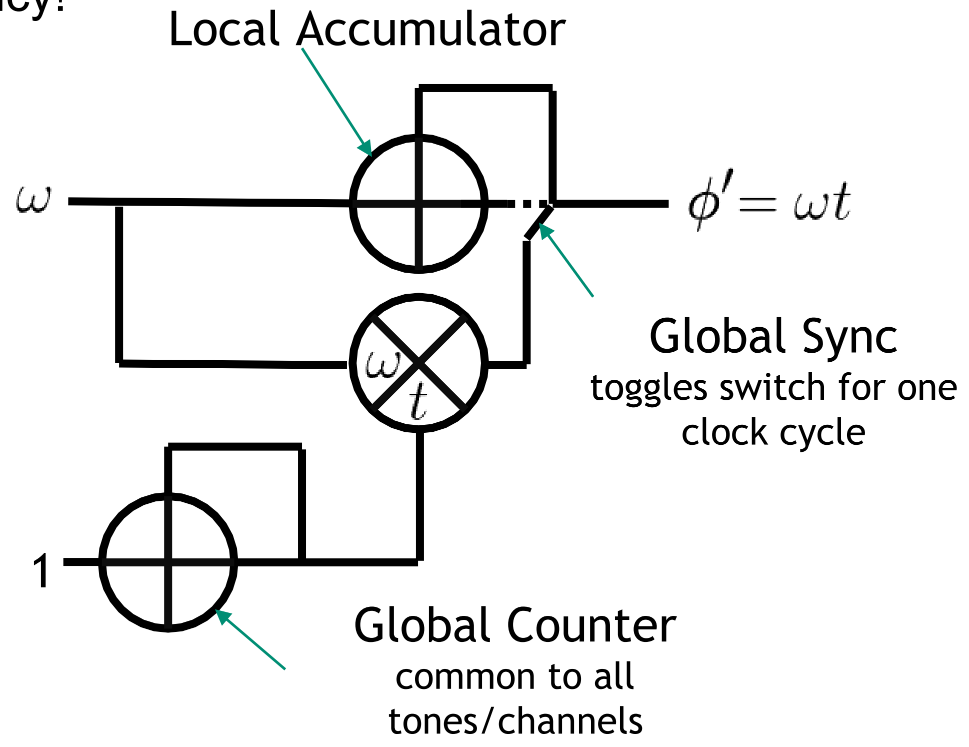
Solution: Let the hardware do the work

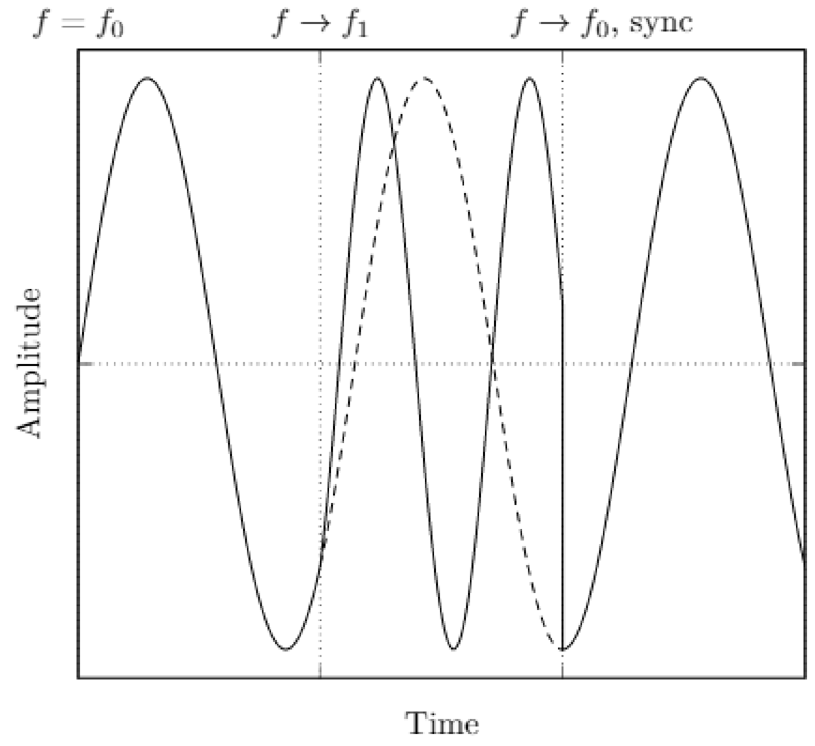
Phases are automatically calculated based on a global counter for the input frequency word

A single pulse can be used to synchronize a tone to match the phase to a point in the past

- As long as its synchronized for the first application at the frequency!





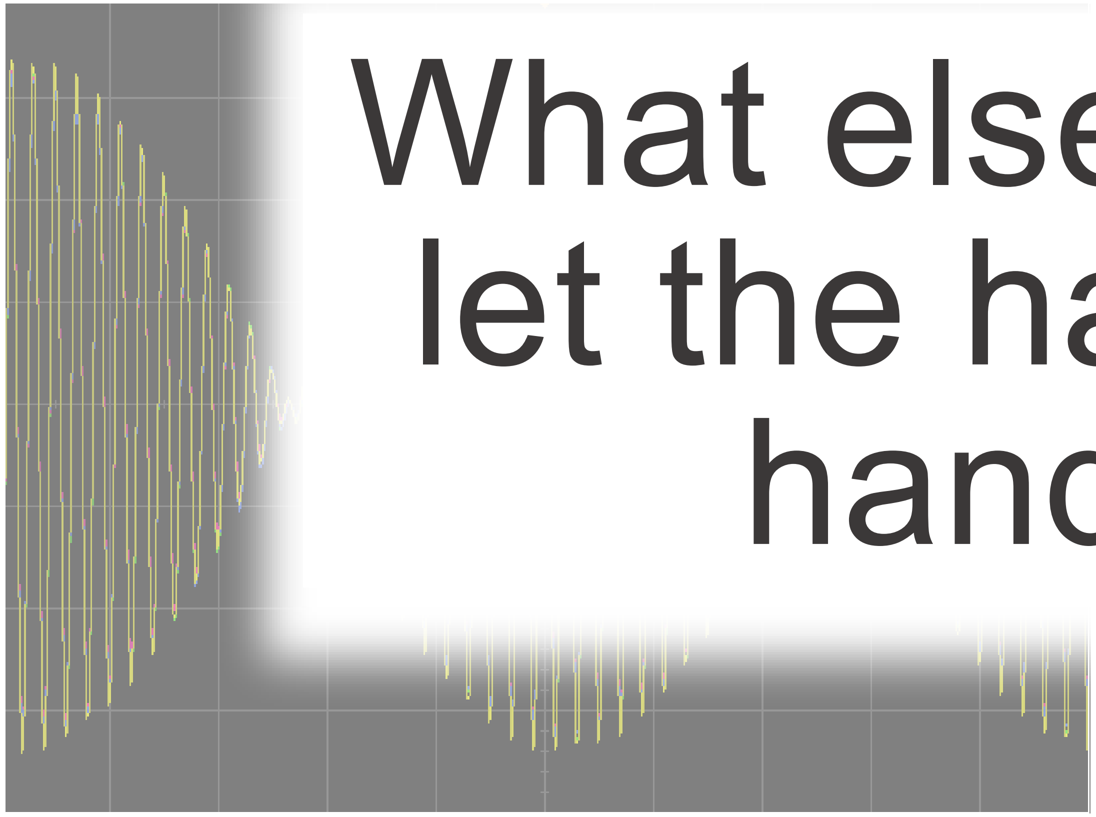


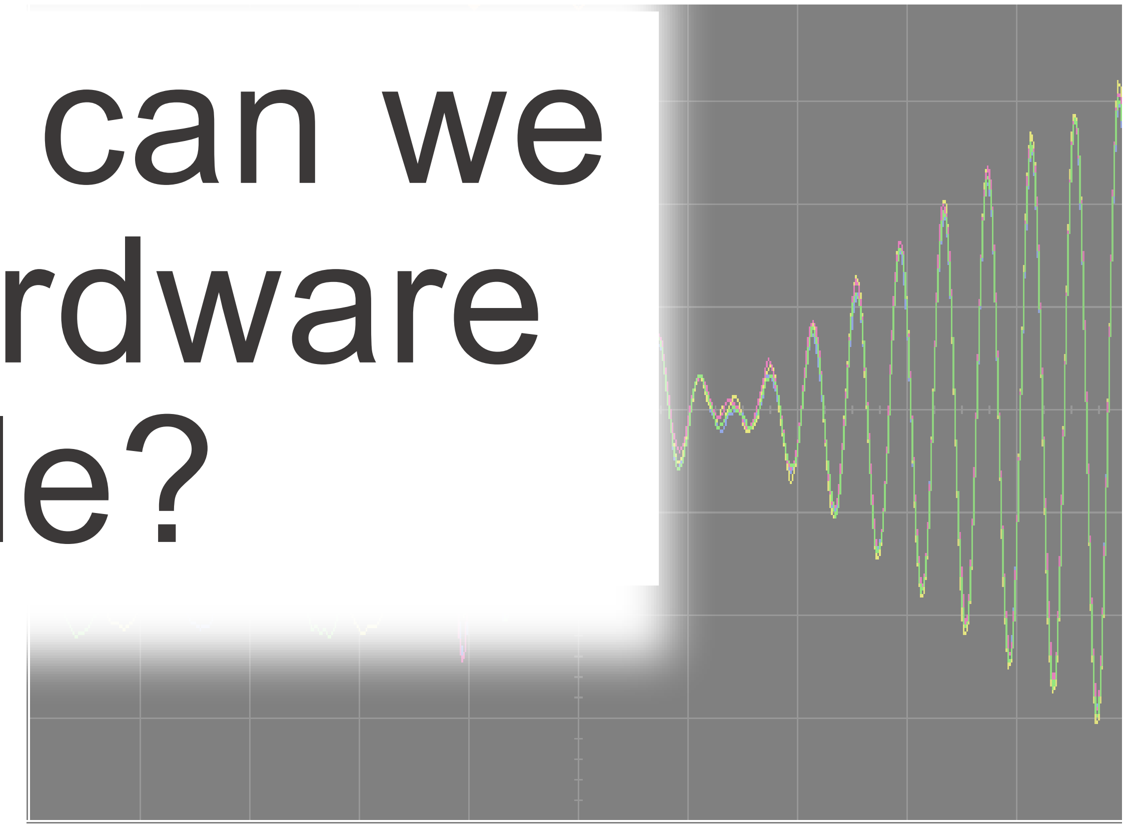
## Hardware Features: Global Phase Synchronization

QSCOUT

Tones being manually synchronized to an earlier state

Changing parameters while applying synchronization





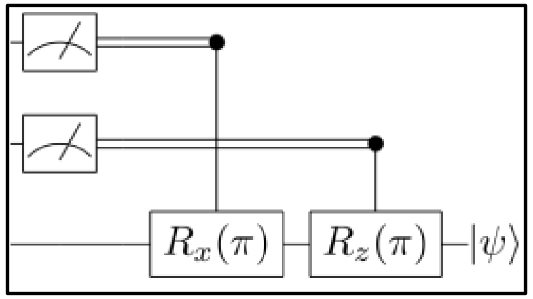
What else can we let the hardware handle?

## Virtual Gates

Without direct access to certain dimensions (e.g. Z), physical gate operations are restricted (e.g. X and Y gates)

Gate sequences such as XZX can artificially represented as YX

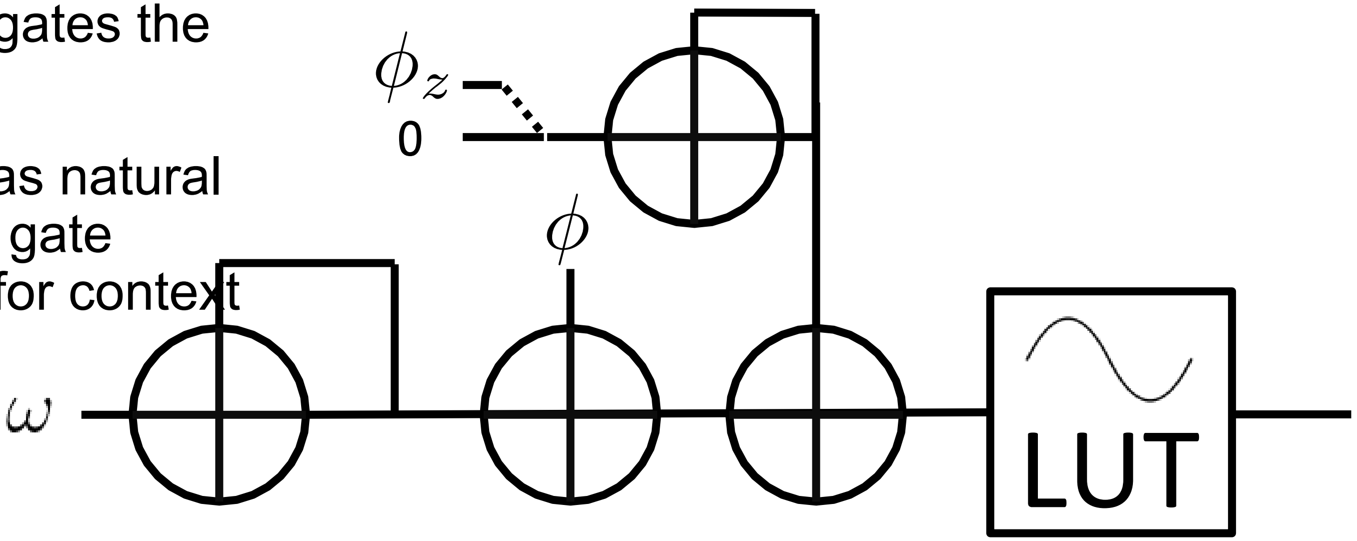
But this is also strongly affected by context dependence!



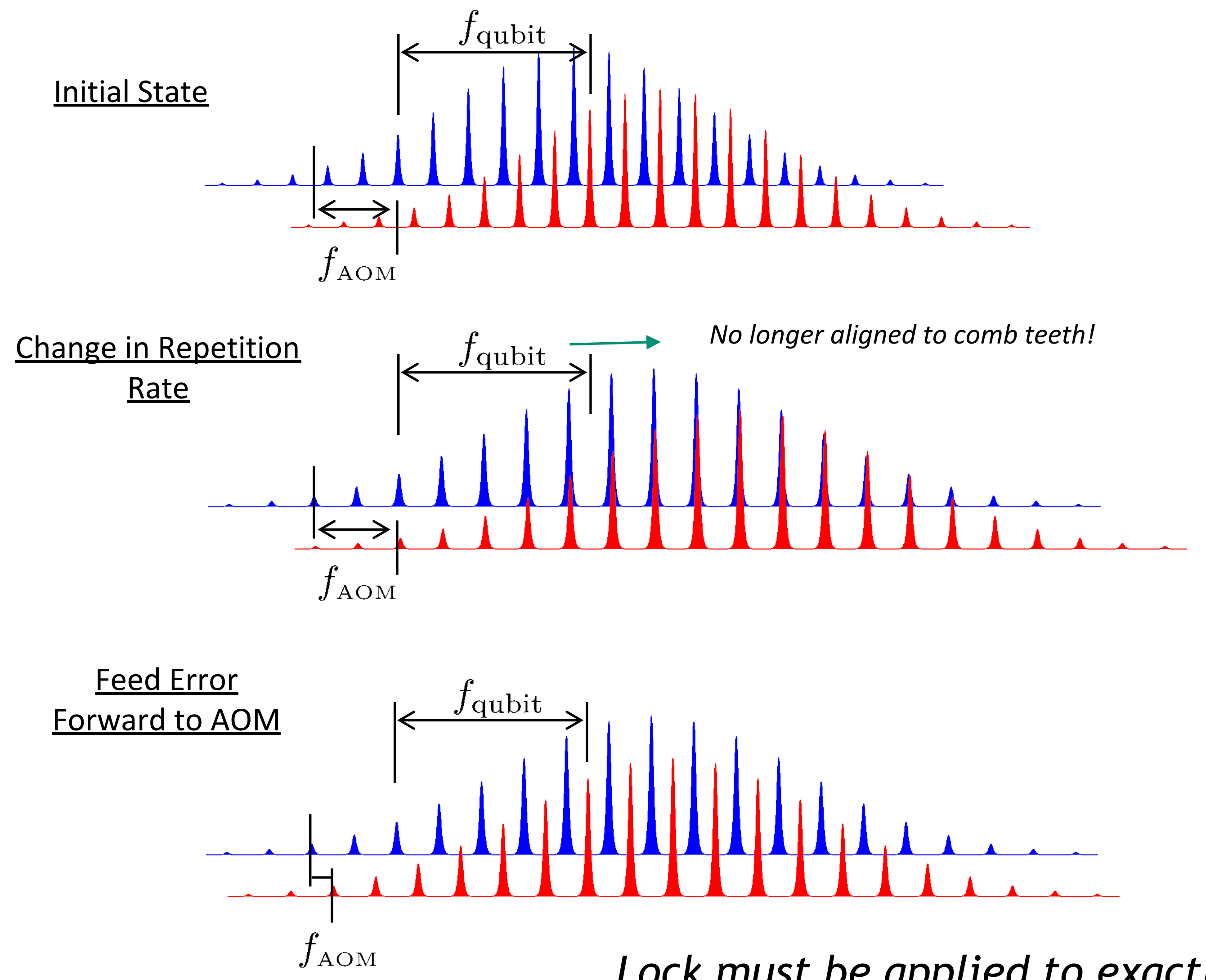
Virtual phase can be independently tracked in an accumulator such that Z gates will automatically add the accumulated phase for all gates the follow

This means that Z gates can exist as natural primitives without having to mutate gate sequences in a way that accounts for context dependence

No manual bookkeeping required!

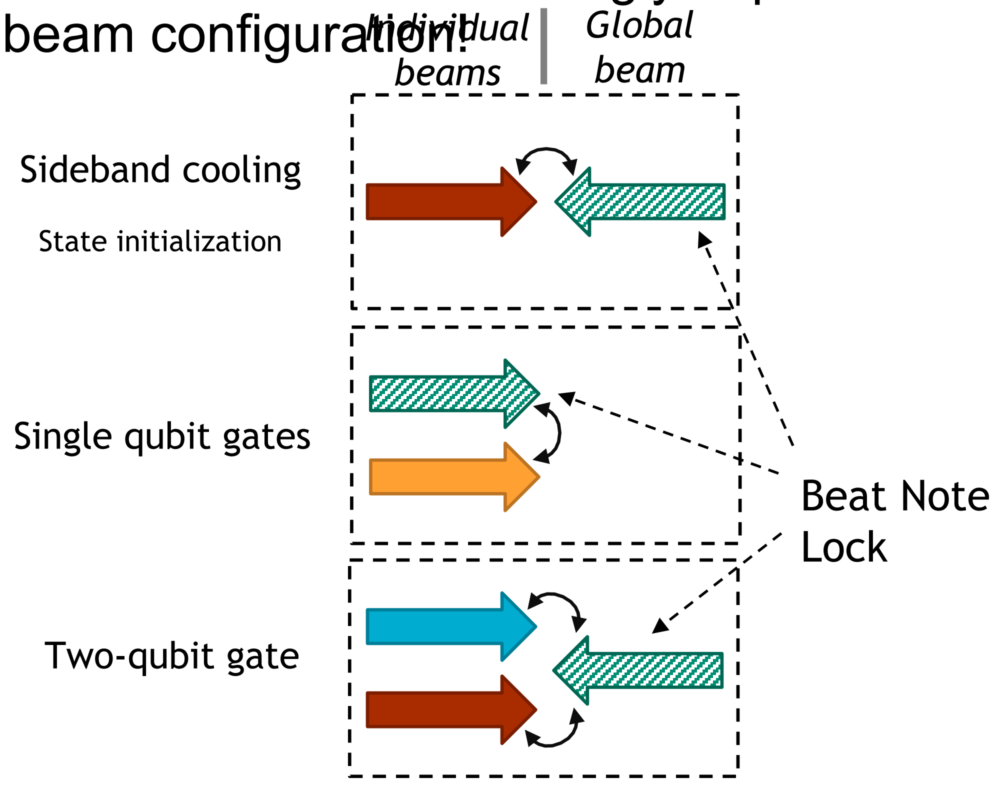


## Dealing with Experimental Imperfections: Cavity Drift



Cavity drift in our pulse laser requires a frequency feedforward lock to correct for errors

How this error is forwarded strongly depends on the beam configuration!



*Lock must be applied to exactly one tone for each Raman pair!*

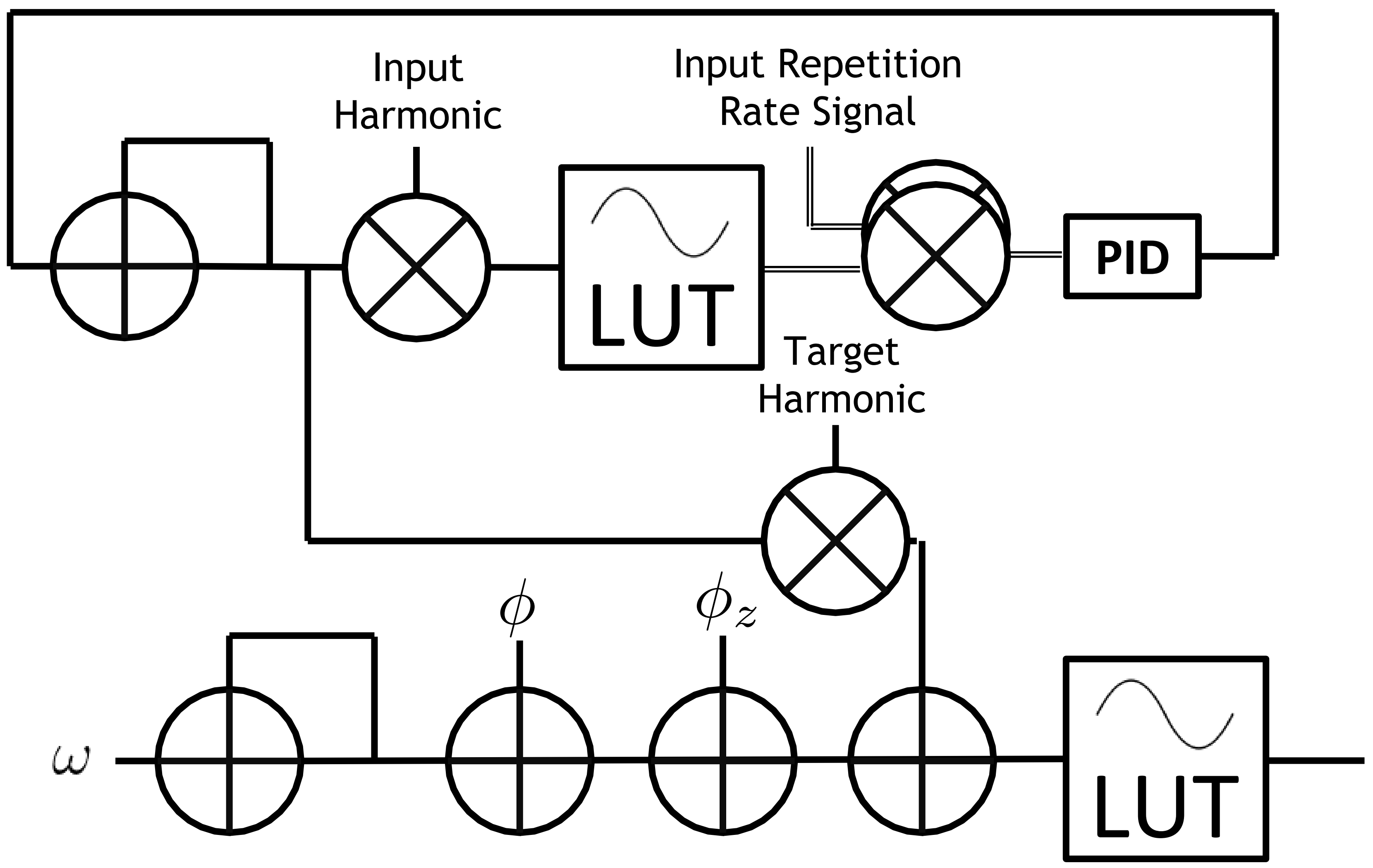
## Dealing with Experimental Imperfections: Cavity Drift

Cavity drift is monitored by measuring output pulses on a photodiode and subsequently mixed down by ~3.7 GHz

The resulting signal is further mixed down via a complex mixer and compared with a dedicated DDS for feedback

By using a multiplier in the feedback loop, we can divide the repetition rate by the input harmonic being measured

The output is then multiplied by the target harmonic, which can be individually configured for each tone and added to the local accumulator output for a lightweight firmware footprint



## Dealing with Experimental Imperfections: Crosstalk Compensation

Different types of crosstalk can occur

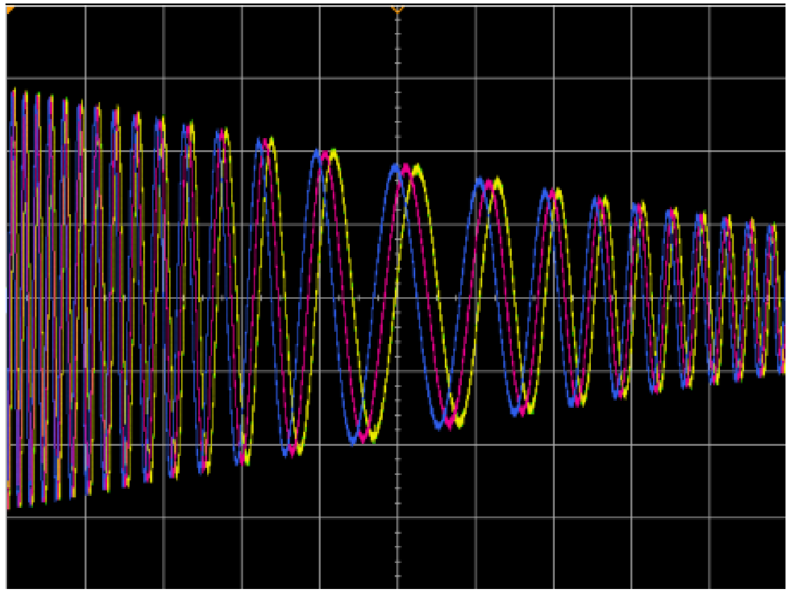
- Optical: overlap of individual addressing beams
- Acoustic: sympathetic vibrations of neighboring crystals in multi-channel AOM
- Electrical: control signal crosstalk on next-nearest neighbors in multi-channel AOM

Want the ability to handle all types!

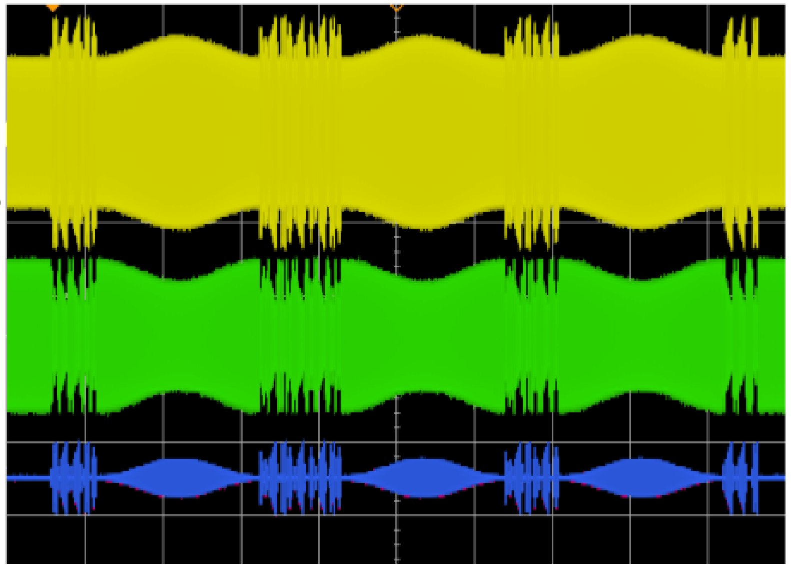
Complex output of each DDS is given an configurable coarse delay and passed to nearest- and next-nearest-neighbor channels

Each input signal is multiplied against a complex amplitude to change the amplitude and phase for fine-tuning alignment

Fine-tuning using phase adjusts



Applying a signal with equal/opposite amplitude on neighboring channels



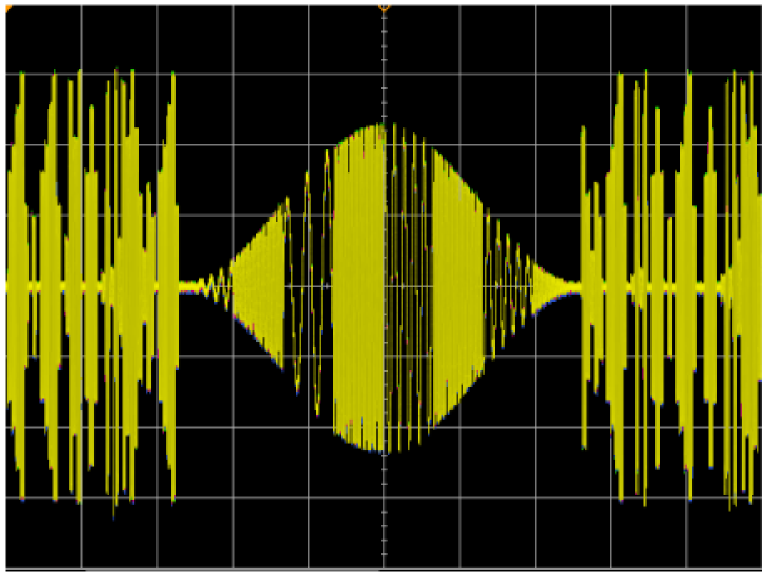
## Quantum Gates

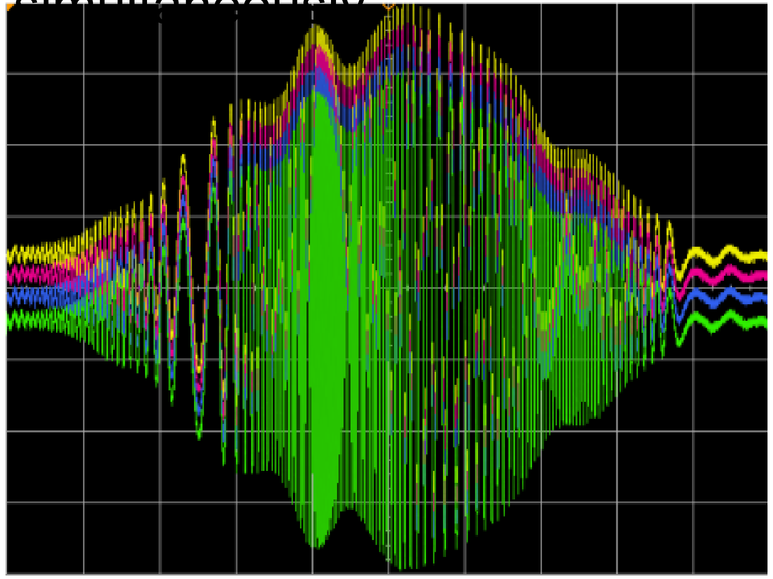
Gates must be defined as discrete “pulses” with precise timing and characteristics to achieve the desired results.

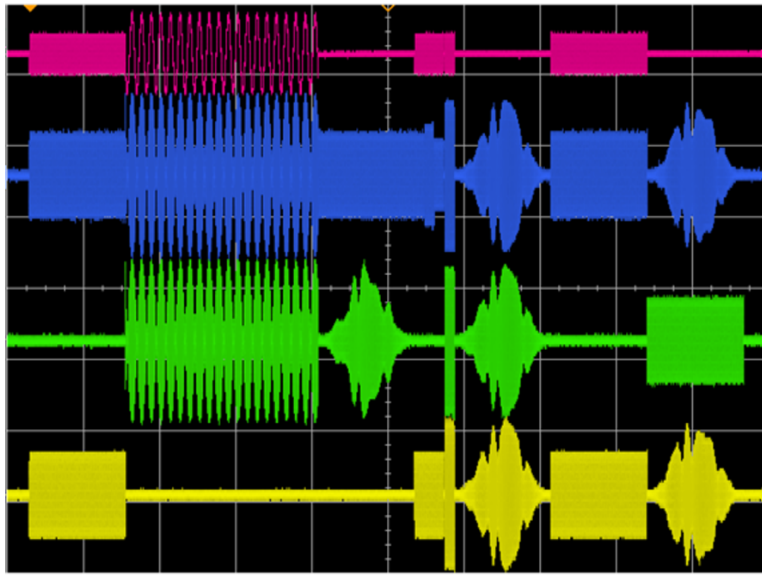
State of the art gate designs require discrete or continuous modulation of frequency, phase, and amplitude.

Gates must be synchronous across all channels and tones, with the ability to run all modulation types simultaneously

Long sequences can be necessary, so a compact representation is needed.







Gate times (2-200 us) are typically much slower than the period (5 ns) of the natural frequencies (200 MHz) needed to drive the AOMs

Instead of writing raw waveform data like an arbitrary waveform generator (AWG) we can advantage of more compact representations

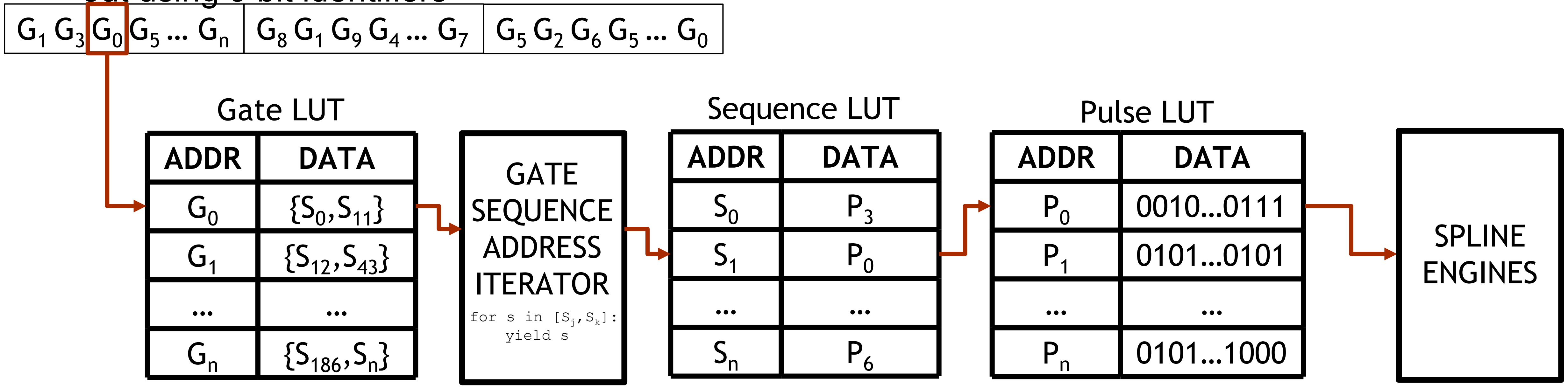
## Gate Sequencer

Gates are represented as natural cubic splines (square pulses force the higher order coefficients to zero) and are interpolated on-chip with update rates of ~2.5 ns

Gate data is often redundant, for example X & Y gates differ only by a phase

Raw gate data is compressed to unique elements and then sequenced on chip using a series of LUTs

The simplest gates require at least 2048 bits of information, but can be stored once and streamed out using 9 bit identifiers



$G_1\ G_3\ G_0\ G_5 \dots G_n$
$G_8\ G_1\ G_9\ G_4 \dots G_7$
$G_5\ G_2\ G_6\ G_5 \dots G_0$
Gate LUT
ADDR | DATA
$G_0$ | $\{S_0, S_{11}\}$
$G_1$ | $\{S_{12}, S_{43}\}$
… | …
$G_n$ | $\{S_{186}, S_n\}$
GATE SEQUENCE ADDRESS ITERATOR
for s in $[S_j, S_k]$: yield s
Sequence LUT
ADDR | DATA
$S_0$ | $P_3$
$S_1$ | $P_0$
… | …
$S_n$ | $P_6$
Pulse LUT
ADDR | DATA
$P_0$ | 0010…0111
$P_1$ | 0101…0101
… | …
$P_n$ | 0101…1000
SPLINE ENGINES

## Gate Sequencer

Compressed representations allow arbitrarily long gate sequences and multiple sequences can be queued up and run back to back with dynamic reprogramming

Compressed representations allow for running large numbers of gates which can be streamed in externally

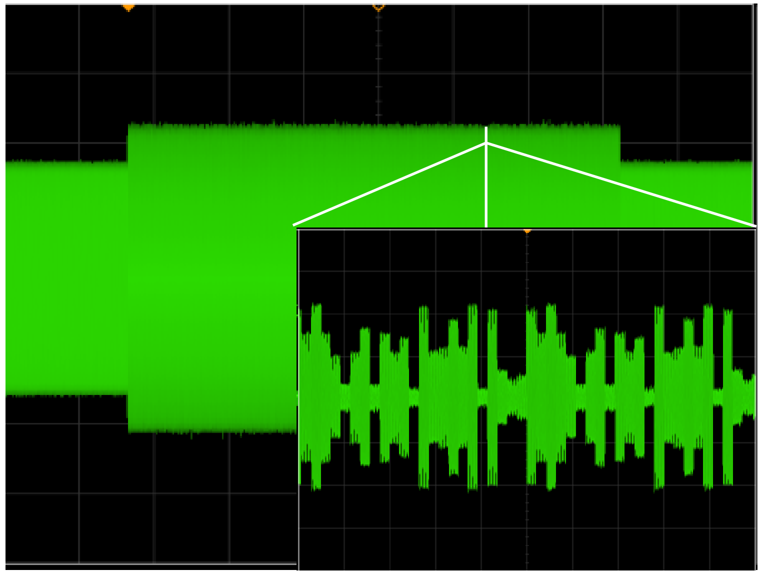
12,000,000 gates each with 100 ns duration (by no means a limit of the system)

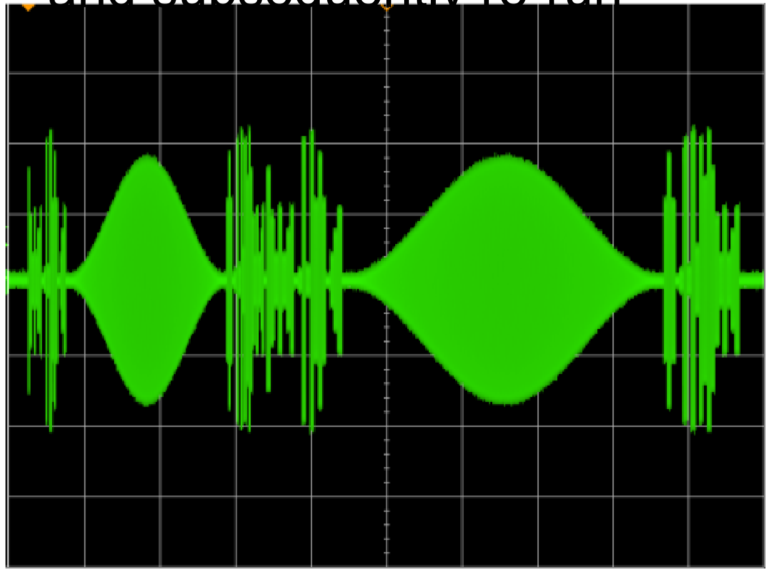
The gate sequencer can be dynamically reprogrammed between gate sequences.

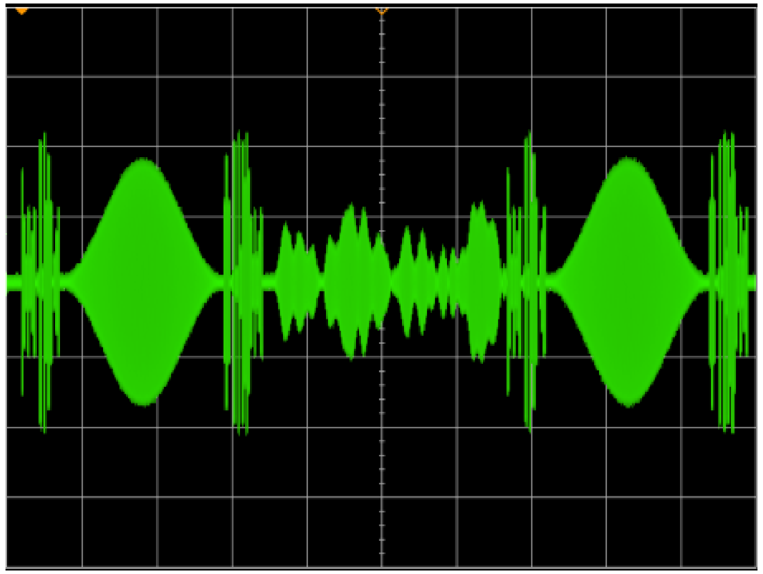
Here the sequencer was run and immediately reprogrammed such that all gates had twice the duration and subsequently re-run

Direct data streaming is supported for one-off sequences.

Though partial re-programming is often the better option

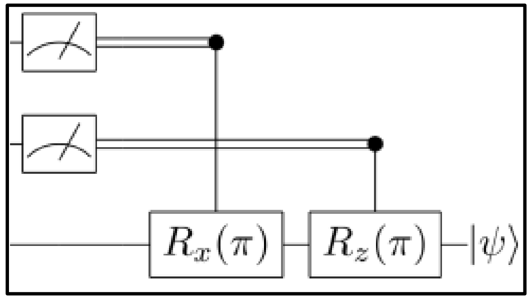






## Fast Branching For Conditional Gate Sequences

When mid-circuit measurements require a conditional sequence of gates to be run, the hardware must be able to react quickly



For situations where these gates are known in advance, they can be passed to the hardware with a partial gate identifier (i.e. address for the gate LUT)

Given a gate identifier of **0b001010** and a measurement result of **0b0011**, the lookup value of the gate address is converted to **0b0011001010** using a matrix-style bitmask

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

**Once a measurement result is complete, a secondary trigger is sent to the gate sequencer such that the additional latency only depends on the latency imposed by the measurement process and extra trigger**

Moreover, depending on how the gate LUT is programmed, one can optionally and dynamically configure the aspect ratio of the matrix lookup, since measurement result masks are simply OR'd with the input gate identifier

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

Since gate identifiers are packed into 256-bit words, multiple gates can be applied based on a single measurement result and chained together across multiple 256-bit words to realize long measurement-based sequences

## Jaqal: Just another quantum assembly language”

Example Jaqal Code

```
from QSCOUT.std.v1 usepulses *

register q[8]

let pi4   0.78539816339
let mpi2 -1.57079632679

macro Hadamard target {
  Sy target
  Px target
}

macro CNOT control target {
  Sy control
  MS control target 0 pi4
  < Rx control mpi2 | Rx target mpi2 >
  Sy control
}

prepare_all
Hadamard q[1]
CNOT q[2] q[1]
measure_all
```

### Features

- Simple interface, easy to learn
- Provides a natural way to write gates that can be run in sequence or in parallel
- Basic elements such as parameter definitions, macros, loops, and qubit aliases to ease programming while maintaining readability and explicitness
- Quickly switch between low-level gate definitions

# JaqalPaw: Jaqal “Pulses and waveforms”

## Features

- Uses Python for flexibility
- Pulse representation is a simple data structure
- Modulation expressed as tuples of spline knots or lists of discrete values
- Splines can be specified for multiple parameters simultaneously an with different lengths

## Example JaqalPaw Code

```
def gate_R(self, qubit, theta, phi):
    phase = (phi < 0)*1 + 0*180 + theta + 0/math.pi*180
    calibrated_rabi = self.single_qubit_rabi_cal[qubit]
    symmetric_amp = 0.5 * self.maximum_amplitude
    duration = self.duration_from_rabi_angle(phi ,
                                             symmetric_amp,
                                             calibrated_rabi)
    lower_frequency = self.adjusted_carrier_splitting/ 2
    upper_frequency = -self.adjusted_carrier_splitting/ 2
    gauss_amp = np.sqrt( self.gauss(7,
                                   symmetric_amp,
                                   freqwidth=200e3,
                                   total_duration=4e-6))
    return [PulseData(qubit,
                      duration,
                      amp0=Spline(gauss_amp),
                      amp1=Spline(gauss_amp),
                      freq0=lower_frequency,
                      freq1=upper_frequency,
                      phase0=0,
                      phase1=phase,
                      fb_enable_mask=0b01,
                      sync_mask=0b11)]
```

## Future Directions

Take full advantage of the hardcore processing systems on the chip

APU: Application Processing Unit for intermediate-level feedback involving gate mutation high-level algorithmic control over gate sequences

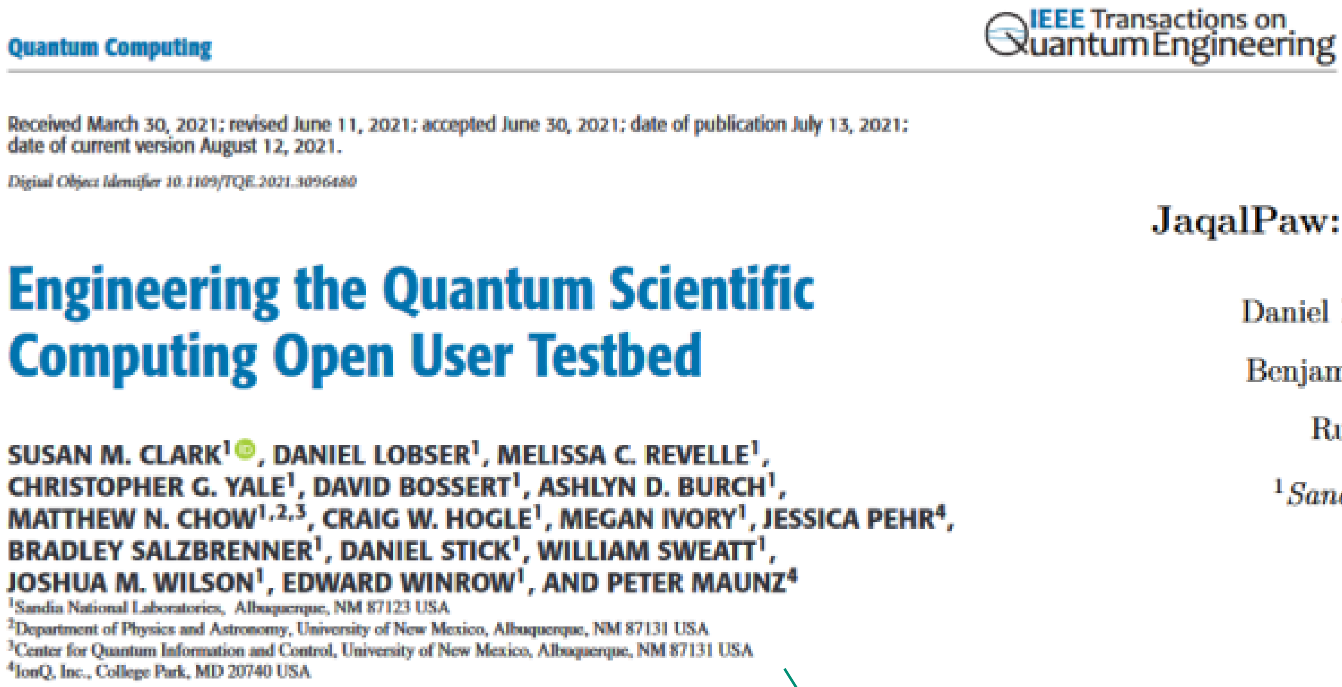
RPU: Real-time Processing Unit for deterministic timing and precise control flow across multiple systems

Adapting MPSoC system as master control hardware for various subsystems, classical control electronics, data handling, and more

## More Resources

QSCOUT

## References



Quantum Computing

IEEE Transactions on Quantum Engineering

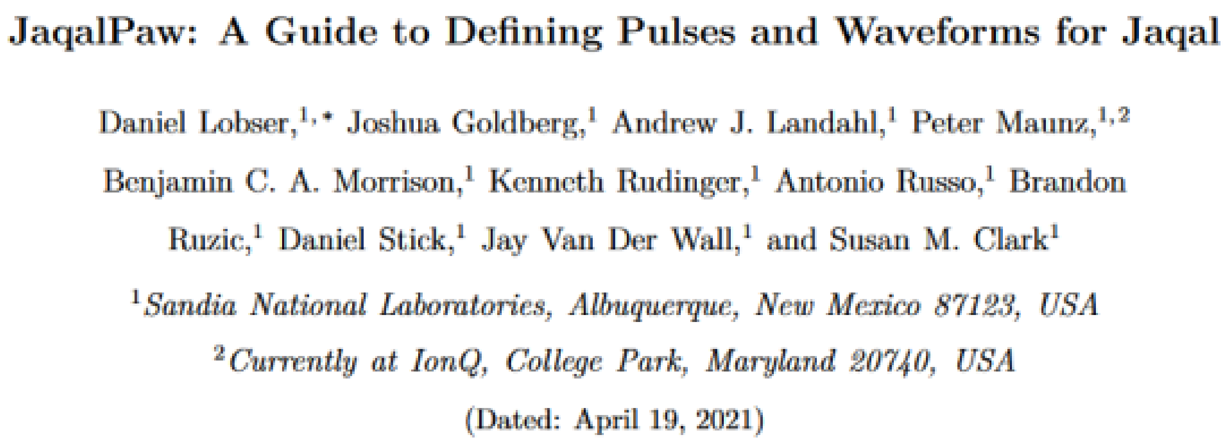
Received March 30, 2021; revised June 11, 2021; accepted June 30, 2021; date of publication July 13, 2021; date of current version August 12, 2021.

Digital Object Identifier 10.1109/TQE.2021.3096480

**Engineering the Quantum Scientific Computing Open User Testbed**

SUSAN M. CLARK[1], DANIEL LOBSER[1], MELISSA C. REVELLE[1], CHRISTOPHER G. YALE[1], DAVID BOSSERT[1], ASHLYN D. BURCH[1], MATTHEW N. CHOW[1,2,3], CRAIG W. HOGLE[1], MEGAN IVORY[1], JESSICA PEHR[4], BRADLEY SALZBRENNER[1], DANIEL STICK[1], WILLIAM SWEATT[1], JOSHUA M. WILSON[1], EDWARD WINROW[1], AND PETER MAUNZ[4]

[1]Sandia National Laboratories, Albuquerque, NM 87123 USA
[2]Department of Physics and Astronomy, University of New Mexico, Albuquerque, NM 87131 USA
[3]Center for Quantum Information and Control, University of New Mexico, Albuquerque, NM 87131 USA
[4]IonQ, Inc., College Park, MD 20740 USA



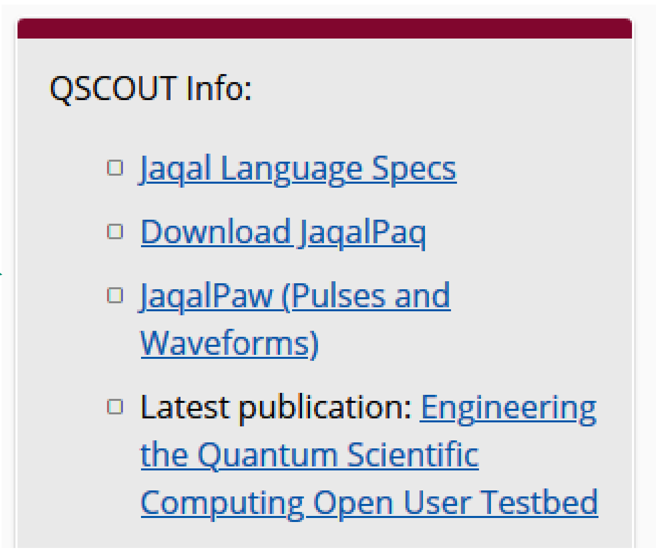
**JaqalPaw: A Guide to Defining Pulses and Waveforms for Jaqal**

Daniel Lobser,[1,*] Joshua Goldberg,[1] Andrew J. Landahl,[1] Peter Maunz,[1,2] Benjamin C. A. Morrison,[1] Kenneth Rudinger,[1] Antonio Russo,[1] Brandon Ruzic,[1] Daniel Stick,[1] Jay Van Der Wall,[1] and Susan M. Clark[1]

[1]*Sandia National Laboratories, Albuquerque, New Mexico 87123, USA*
[2]*Currently at IonQ, College Park, Maryland 20740, USA*

(Dated: April 19, 2021)

https://qscout.sandia.gov



QSCOUT Info:

- Jaqal Language Specs
- Download JaqalPaq
- JaqalPaw (Pulses and Waveforms)
- Latest publication: Engineering the Quantum Scientific Computing Open User Testbed

## Questions?

Thanks to our team and collaborators

**Peter Maunz**
Susan Clark
Ashlyn Burch
Matt Chow
Craig Hogle
Megan Ivory
Melissa Revelle
Dan Stick
Andrew Van Horn
Josh Wilson
Chris Yale

**Jay Van Der Wall**
**Josh Goldberg**
Andrew Landahl
Ben Morrison
Tim Proctor
Kenny Rudinger
Antonio Russo
Brandon Ruzic
Kevin Young
Collin Epstein
Andrew Van Horn

Brad Salzbrenner
Madelyn Kosednar
Jessica Pehr
Ted Winrow
Bill Sweatt
Dave Bossert

Matt Blain
Ed Heller
Jason Dominguez
Chris Nordquist
Ray Haltli
Tipp Jennings
Ben Thurston
Corrie Sadler
Becky Loviza
John Rembetski
Eric Ou
Matt Delaney

UNM
**Nafis Irtija**
**Jim Plousquellic**
Eirini Tsiropolou

Duke
Ken Brown
**Marko Cetina**
Jungsang Kim
Chris Monroe

AOSense
**Alan Bell**