# Developing a Quantum Step-sequencer

**María Aguado Yáñez**[1]

[1]Music Technology Group – Pompeu Fabra University
Roc Boronat, 138 - 08018 - Barcelona - Spain

maria.aguado02@estudiant.upf.edu

***Abstract.*** *As quantum computing continues to advance, there is a growing interest in exploring its applications in the field of music technology. This paper presents a step-sequencer that utilizes quantum algorithms for music generation. The step-sequencer employs a Launchpad X interface to control the generation of MIDI notes, which are then processed in Python. The resulting commands are sent to either a Pd patch or the DAW Cubase for sound synthesis. The interface consists of an eight by eight grid of buttons, with the first three rows associated with qubits. Different gates can be manually chosen and applied to these qubits, allowing for intuitive manipulation of quantum systems. The step-sequencer divides the Launchpad X into two halves, with each half representing eight steps. While the qubits operate under quantum principles, the fourth row of the interface is dedicated to chords and bass as "classical instruments". This combination of quantum and classical instruments enables the generation of both steady and repetitive sequences, as well as counterintuitive and surprising musical patterns. The paper provides detailed explanations of the interface, code, and sound synthesis components, highlighting the functionality and capabilities of the quantum step-sequencer.*

## 1 Introduction

As the field of quantum computing continues to expand, there is a growing interest in exploring its applications within the realm of music technology. Fundamental principles of quantum mechanics, such as superposition and entanglement, are gaining popularity, and there is a demand for the development of musical devices that offer intuitive ways of manipulating quantum systems. This paper introduces a step-sequencer that serves the purpose of showcasing the use of quantum algorithms for music generation.

A step-sequencer is a musical device or software tool designed to create and manipulate musical patterns in real-time. It is widely used in electronic music production and live performances. A step-sequencer organizes musical sequences into discrete steps, where each step represents a specific point in time or beat. By programming the steps with different musical events or notes, composers and performers can create rhythmic patterns, melodic sequences, and intricate arrangements.

This step-sequencer harnesses quantum principles to produce music by aggregating qubits, thereby introducing a new layer of abstraction to the creative process. It is important to mention that the step-sequencer does not use real quantum qubits, but a simulation of them on a classical computer. Further updates of the code may include the use of true quantum random numbers rather than deterministic classical numbers.

The inspiration to develop this device was drawn from chapters 1 to 7 of the book *Quantum Computer Music: Foundations, Methods and Advanced Concepts* [1], which provide an introduction to the generation of music using quantum algorithms.

In Section 2, an explanation of each component of the step-sequencer is presented. These components consist of the interface, the code, and the sound synthesis, which collectively contribute to the functioning of the step-sequencer.

## 2 The Quantum Step-Sequencer

The quantum step-sequencer consists of a sixteen-step grid, representing semiquavers. The interface used for this purpose is the Launchpad X, which sends MIDI notes to the computer. The information is then processed in Python, and the commands are transmitted to either a Pd patch or the DAW Cubase, where sound synthesis takes place.

### 2.1 The interface: Launchpad X

A quantum circuit is associated with an eight by eight grid of buttons of a Launchpad X device. Fig.1 shows the mapping of the different instruments to the buttons.

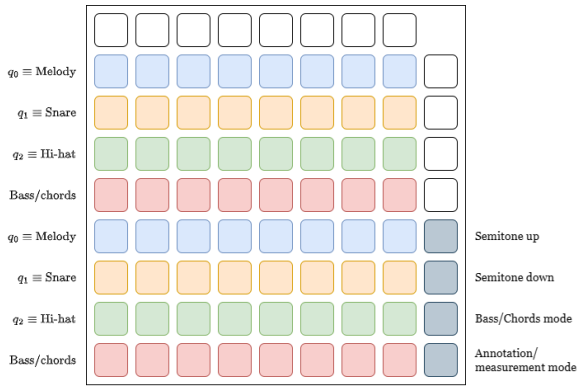


**Figure 1: Interface mapping**

The first three rows are associated with three qubits. On each square of a qubit row, different gates are mapped. Each time a qubit button is pressed it will be applied a different type of gate, and this gate will be maintained until it is pressed again. Thus, pressing repeatedly a qubit button iterates through different gates. When a button is pressed enough times to end the gate-loop iteration, it starts over again. These qubits can be measured by activating them using toggle buttons on the measure mode,

which can be accessed by pressing the key in the right corner. Only the activated qubits will be measured when the step-sequencer is running. The first qubit, $q_0$, generates the melody. The other two qubits, $q_1$ and $q_2$, are associated with a snare and a hi-hat. There exists a distinction between $q_0$ and $q_1$, $q_2$ in the measure mode: meanwhile $q_0$ is ***annotated***, $q_1$ and $q_2$ are ***measured***. This difference will be explained in section 2.2.2.

The fourth row is associated with chords and a bass (each button is a note within an octave). Pressing the button on the right corner below allows the change of mode from chords to bass and vice versa. Each of the squares of the chords/bass row is associated with a note frequency.

Since there are sixteen steps, the Launchpad is divided into two halves of eight steps. The structure of the first half is repeated on the second one. Notice that only the three qubits work under quantum principles, whereas the chords and the bass are "classical instruments". Since quantum-generated music sounds unpredictable, a combination of quantum and classical instruments helps generate a sequence both steady and repetitive (thanks to the classical instruments), and counterintuitive and surprising (quantum instruments).

Additionally, the interface includes two buttons at the bottom-right corner that allow transposing the scale of the melody one semitone up or down. Further buttons could be programmed to include additional transpositions, such as octave transpositions.

All the actions performed on the interface trigger a corresponding light response. The MIDI values for the light responses were determined by referencing the user's manual of the Launchpad X device [2].

## 2.2 The code: Python notebooks

The code is constituted by four notebooks: circuit.ipynb, launchpad.ipynb, pdpython.ipynb and cubpython.ipynb. The first one contains the functions that create the circuits with their specified gates and define the measure/annotation mode using the Qiskit library [3]. The second one computes the MIDI information arriving from the Launchpad and applies the functions in circuit.ipynb. Lastly, the two remaining notebooks establish the connection either with Pd or Cubase and send the information computed in launchpad.ipynb. The connection with Pd and Cubase is unidirectional since the information is only wanted to be sent from Python to Pd or Cubase and not the other way around.

Since the circuit.ipynb notebook describes the quantum processes inside the step-sequencer, it is explained in more detail below.

### 2.2.1 Gates

A function defined in the circuit.ipynb notebook is in charge of generating the circuits (see Fig.2). The gates that these circuits are going to be using are selected through the Launchpad interface. Each time a qubit button is pressed, a loop iterates changing the gate. The gates used in this step-sequencer are:

1. Identity gate (I): it leaves a qubit invariant when it is applied.
2. Hadamard gate (H): it induces an equally-weighted superposition of two opposing states.
3. Not gate (X): it rotates a qubit in the Bloch's sphere to its opposite position (180º degrees rotation).
4. Rotation 1 around z-axis ($\pi/4$ up: $R_{up}$): it rotates the qubit $+\pi/4$ around the z-axis.
5. Rotation 2 around z-axis ($\pi/4$ down: $R_{down}$): it rotates the qubit $-\pi/4$ around the z-axis.
6. Conditional not gate (CX): it executes an X gate on a qubit under a certain condition imposed on another qubit. Since the state of one qubit affects a second one, a CX gate induces entanglement.

To apply and CX gate between two qubits, the first one should be held in the Launchpad while pressing the second qubit button. This is the only gate that is not encoded in the loop iteration, and it is only activated under the condition of a qubit button being held.

```
#gate labels: 0 = I, 1 = H, 2 = X, 3 = Rup, 4 = Rdown, 5 = CX1, 6 = CX2

def createCircuit(data): #function that creates a circuit using the gates selected in the Launchpad.
    circ = qk.QuantumCircuit(3)
    for i in range(3):
        #add X gates
        if data[i] == 2: #the data is a list of 3 integers, each one representing a qubit. The integers are the gate labels.
            circ.x(i)
        #add H gates
        if data[i] == 1:
            circ.h(i)
        #add CX gates. The CX gate is a two-qubit gate, so we need to search for the other qubit.
        if data[i] == 5:
            #search for the other qubit
            for j in range(3):
                if data[j] == 6:
                    circ.cx(i, j)
                    break

        #add rotation gates. Gate Rz is a rotation around the z axis (vertical axis). That's phi in the Bloch sphere.
        if data[i] == 3: #rotation gate for the up state
            circ.rz(kR, i)
        if data[i] == 4: #rotation gate for the down state
            circ.rz(-kR, i)
    return circ
```

**Figure 2: Function for the circuits**

### 2.2.2 Measurements/Annotations

Another function defined in the circuit.ipynb notebook handles the measure/annotation mode. Each qubit is activated on a step if its corresponding button in the measure/annotation mode is pressed. If a qubit button is not pressed, the qubit is not measured/annotated.

The first qubit, $q_0$, generates the melody. The notes of the F# minor scale are mapped to angles around the z-axis in this qubit's Bloch sphere (see Fig.3). When measuring the qubit, the information about its phase in the Bloch sphere is lost due to wave function collapse into a single state. Hence, instead of measuring, this qubit should be annotated. On the other hand, qubits $q_1$ and $q_2$ are measured. These qubits represent two different timbres of snare and hi-hat, respectively. Therefore, measurement is necessary to ensure that only one timbre of snare and one timbre of hi-hat are played at a time (see equations 1 and 2).

$$|\psi_{snare}\rangle = a\,|snare_1\rangle + b\,|snare_2\rangle. \qquad (1)$$

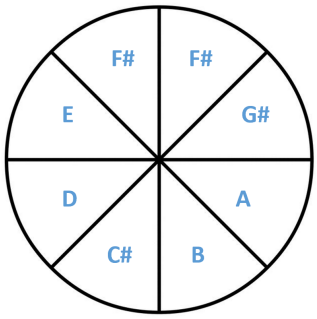


**Figure 3: Notes associated to each angle**

$$|\psi_{hihat}\rangle = a\,|hihat_1\rangle + b\,|hihat_2\rangle. \tag{2}$$

It is important to note that when annotating $q_0$, qubits $q_1$ and $q_2$ must always be measured, regardless of whether their measurement is desired or not. This is because $q_1$ and $q_2$ could be entangled with $q_0$, and entanglement induces hyperspheres where annotating Bloch's angles is no longer possible unless $q_1$ and $q_2$ are collapsed. When they collapse, the states of $q_0$ can be described again with a Bloch sphere, as shown in the following equation

$$|\psi_0\rangle = a, |0M_1M_2\rangle + b, |1M_1M_2\rangle, \tag{3}$$

where $M_1$ and $M_2$ are the results of the measurements of $q_1$ and $q_2$, respectively. Thus, even if their corresponding light-buttons on the interface are not activated, they are still measured. This information is not used to generate sound but to reduce the dimension of the hypersphere. If their lights are on, sound will be created as well.

### 2.3 Sound synthesis

There are various options available for generating sound in the context of this step-sequencer. Initially, the design focused on sending MIDI notes to Pure Data (Pd) for its simplicity. However, to facilitate tasks such as applying effects, filtering, mixing, and equalizing, using a Digital Audio Workstation (DAW) like Cubase or Ableton proves to be more convenient. Consequently, a project was created in Cubase 11 to receive MIDI notes from the Python notebooks.

#### 2.3.1 Pd Patch

In the Pd patch depicted in Figure 4, Pd receives values from Python for each instrument (melody, snare, hi-hat, chords, and bass). The melody, chords, and bass are generated using sawtooth waves with applied amplitude envelopes. The sliders shown in Figure 5 control the volume and amplitude envelope parameters for these three synthesized instruments. A four-on-the-floor kick sound is also added in Pd. Unlike other instruments, the kick is not controlled by the Launchpad. The sounds of the kick, snare, and hi-hat are imported samples. For the snare and hi-hat, there are two different sounds available, and the selection is based on the result of the quantum circuit computation.

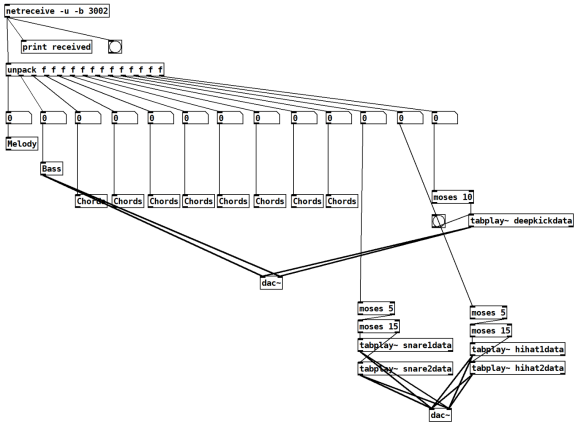


**Figure 4: Pd Patch**

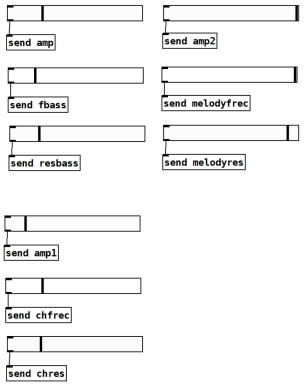


**Figure 5: Pd Patch Sliders**

#### 2.3.2 Cubase

A project was created in Cubase 11 to synthesize the sound, as shown in Fig.6. The project consists of tracks dedicated to each instrument. Unlike the Pd patch, a modification was made to the Python code to send a MIDI value for triggering the kick sound. The snare, hi-hat, and kick are pre-recorded samples imported into Cubase. On the other hand, the melody, chords, and bass are generated using Virtual Studio Technology (VST) plugins.

For each track, a MIDI filter was applied to ensure that the incoming MIDI notes from Python trigger only the assigned instrument. This type of filtering is demonstrated in Fig.7, where it can be seen how the channels and MIDI notes are specified and filtered.

## 3 Conclusions and further work

This paper presented a step-sequencer that incorporates quantum algorithms for music generation. By leveraging the principles of quantum computing, such as superposition and entanglement, the step-sequencer introduces a new dimension to the creative process in music technology. The interface, implemented using the Launchpad X, allows intuitive manipulation of quantum systems through a grid of buttons.

The combination of quantum and classical instruments in the step-sequencer enables the creation of both

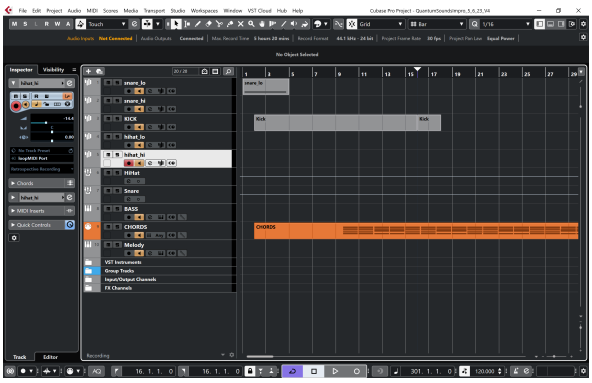

**Figure 6: Cubase Project**

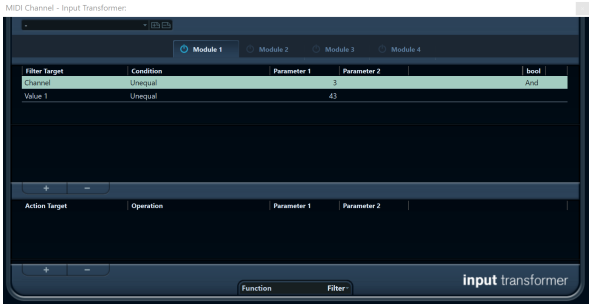


**Figure 7: Channel Filtering**

predictable and repetitive sequences, as well as unexpected and surprising musical patterns. The melody, chords, and bass are generated using quantum principles, while the snare, hi-hat, and kick are triggered by classical instruments. This integration of quantum and classical elements opens up new possibilities for exploring musical composition and expression.

The paper provided detailed explanations of the interface, code, and sound synthesis components, highlighting the functionality and capabilities of the quantum step-sequencer. By presenting the underlying concepts and the technical implementation, this work serves as a foundation for further exploration and experimentation in the field of quantum music technology.

Currently, the step-sequencer utilizes a simulation of quantum qubits on a classical computer. To enhance the authenticity and explore the true potential of quantum music, integrating with real quantum hardware can be pursued. This would require adapting the code and communication protocols to interface with quantum devices.

## References


[1] E.R. Miranda. *Quantum Computer Music: Foundations, Methods and Advanced Concepts*. Springer Nature, 1st edition, 2022.

[2] *Launchpad X: Programmer's reference manual*. Focusrite Audio Engineering Limited, 2019.

[3] Qiskit web page: https://qiskit.org/.