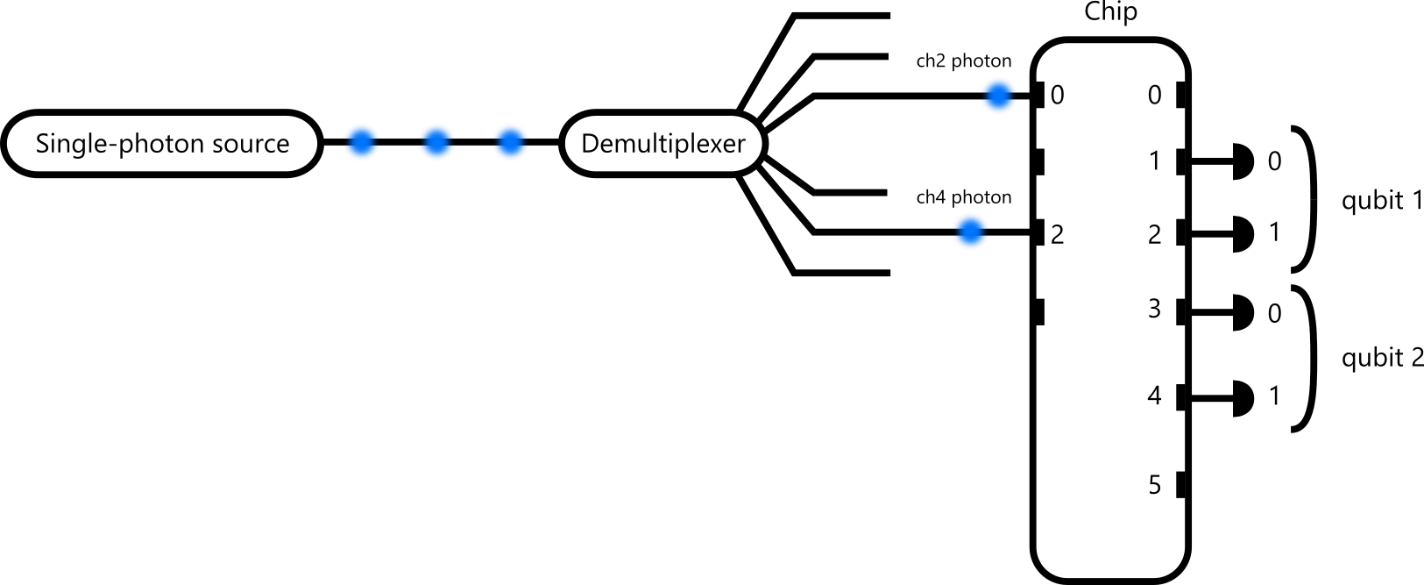
CNOT with delayed photons

We implement a CNOT gate on our 12-mode QPU and study its fidelity as the delay between the two input photons varies from 0 ps to 30 ps.

Photon velocity in optical fiber m/s.

# Experimental setup



Single photons are emitted by a Quandela quantum dot. A demultiplexer sends the photons into different paths and delays (not shown in the figure) align temporally the photons at the chip input. The ch2 photon is delayed by 72 m of fiber and ch4 is delayed by 36 m (ie. the ch2 photon is emitted 180 ns after the ch4 photon by the source). The two photons are injected in inputs 0 and 2 of the 12-mode photonic chip. The chip implements the CNOT gate and we collect the results on outputs 1 to 4 of the chip. Notice that to evaluate the CNOT gate on different 2-qubit input states, we do not change the physical input ports of the photons, but instead the chip applies permutations on the input modes.

To change the delay between the photons, we slide translation stages in the demultiplexer, which changes the optical path of the photons in free space.

# CNOT phases

In the folder “cnot phases”, you can find the phases applied on the photonic chip for each input state of the gate (as stated above, the physical photon input is fixed, but with the chip we can apply permutations on the photons to make them switch modes within the chip). The json files contain a dictionary with following keys:

{

**“phases”**: list[float], values of individual phases in radians

“**components**”: list, name of phases that dictate the order of the “phases” list

}

Without going too much into detail about the name of the phases, the order is basically the following:

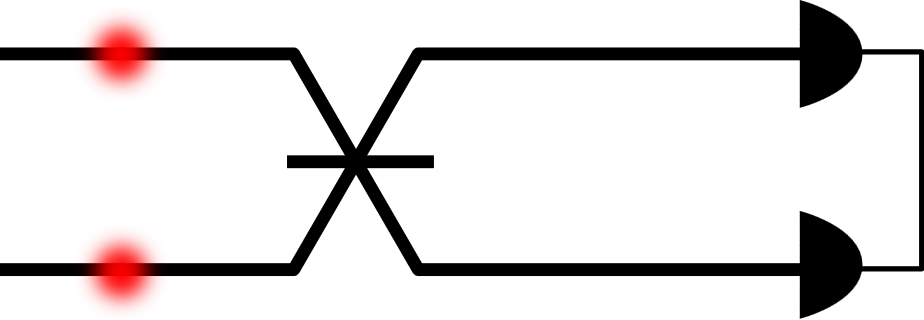
## Chart, scatter chart Description automatically generated

where a blue horizontal bar indicates a phase in a Mach-Zehnder interferometer (BS+phase+BS), and a red vertical bar indicates a phase.

# HOM measurements

The Hong-Ou-Mandel visibility of 2 photons quantifies their indistinguishability in all degrees of freedom (polarization, time of arrival, position, energy, orbital momentum…). When , photons are perfectly indistinguishable, and when , they are completely distinguishable.

It is usually measured with the following circuit:



where the component in the middle is a symmetric beamsplitter. is then given by the formula:

where is the probability of measuring a 2-photon coincidence (two photons detected at the same time, one per detector). To measure on our photonic chip, we set the first two rows of phases in red bars to (green marker in figure below), making them act effectively as symmetric beamsplitters.

Chart, scatter chart

Description automatically generated

In the case of perfectly pure single photons (that is the source emits only one photon at a time), directly measures the photon wavefunction overlap :

But in practice, single-photon sources sometimes emit two photons instead of one. The frequency of this event is quantified by the measurement. As a result, does not immediately give the wavefunction overlap , but this formula takes this into account:

In our case (see section “g2” below), 0.007.

To recap, if you measure , with our value of , the photon wavefunction overlap is .

We stress that all given below in the data folders are **not corrected** by , such that you have access to the rawest data possible.

# Collected data

Folder “g2” contains a measurement (photon purity) for inputs 0 and 2.

The .png file is a picture summarizing the result. The \_data.txt contains the x- and y-data of the plot. The \_metadata.json contains all the data associated with the measurement. It is a dictionary that can be loaded directly in Python with entries:

{

**‘value’**: float, value

**‘peaks’**: list[int], position of peaks as indices in x-data

**‘bounds’**: list[list[int, int]], list of index couples, indicating the integration bounds of each peak in terms of indices in x-data

**‘baseline’**: float, value of noise baseline, here not taken into account

**‘type’**: str, specifies measurement

**‘laser rep’**: float, laser repetition rate in MHz computed from peak positions

**‘integration data’**: list[list[index]], same as ‘bounds’, but this time in terms of delay in ps

**‘integration span’**: str, width of peak integration window

}

Folder ‘hom’ contains an HOM measurement, with the same settings as xp\_00 (that is ps delay between photon arrivals). Folder structure is the same as g2 folder.

Folders xp\_00 to xp\_05 contain the experiment data. They contain an HOM measurement, whose data structure is the same as the g2 measurement described above. Delay.png shows a plot of the measured photon times of arrival from both input channels. The exponential decay of the quantum dot’s lifetime is convoluted with:

* Laser trigger used to set the reference time (we have contacted the laser manufacturer for the value and will put it on the Teams channel as soon as we know)
* Excitation laser pulse duration ( 10 ps)
* Single-photon detector jitter ( 18 ps)
* Time tagger jitter ( ps)

The x-axis is measured in ps (we forgot to label the units for the x-axis, this is very bad practice). The y-axis is normalized such that both peaks reach their maximum at 1.

The data.json file is dictionary containing:

{

**‘delay data’**: {

“ch3”: dict[str] with keys ‘x’ and ‘y’ containing the x- and y-data for the delay.png plot for the curve labelled **ch2 (yes, we mixed up a bit the channel names)**

“ch4”: dict[str] with keys ‘x’ and ‘y’ containing the x- and y-data for the delay.png plot for the curve labelled ch4

“delay”: delay value between photons from ch2 and ch4 in ps. It is computed by fitting an order 2 polynomial on the points from delay.png which satisfy and taking the position of its maximum.

}

**‘hom’**: float, measured HOM value

**‘cnot’**: dict[str], dictionary whose keys correspond to each input 2-qubit state. The dictionary values are also dictionaries, containing the number of 2-photon coincidences measured for each 2-qubit state.

}

Don’t hesitate to send a message if you need anything additional or to clarify something :)