Silicon qubits Delft

This device is based on reference: https://www.nature.com/articles/s41586-022-05117-x

VQD setup

Set the main directory as the current directory

In[100]:=

```
SetDirectory[NotebookDirectory[]];
```

Load the QuESTLink package

One may also use the off-line questlink.m file, change it to the location of the local file

In[101]:=

```
Import["https://qtechtheory.org/questlink.m"]
```

This will download a binary file **quest_link** from the repo; some error will show if the system tries to override the file

Use **CreateLocalQuESTEnv[quest_link_file]** to use the existing binary

In[102]:=

```
CreateDownloadedQuESTEnv[];
```

Load the **VQD** package; must be loaded after QuESTlink is loaded

In[103]:=

```
Get["../vqd.wl"]
```

Set the default configuration of the virtual Silicon device

frequency unit: ***MHz***

time unit: ***μs***

In[104]:=

```
Options[SiliconDelft] =
  {
   QubitNum → 6
   ,
   (* average of T1 *)
   T1 → 10^4
   ,
   (* In practice, T2 is obtained by echo RX[π/2]–τ–RX[π]–τ–RX[π/2], where τ=1μs. We assume the T2* is echoed out to T2 *)
   T2 → <|0 → 14, 1 → 21.1, 2 → 40.1, 3 → 37.2, 4 → 44.7, 5 → 26.7|>
   ,
   (* Qubit frequency of each qubit *)
   QubitFreq → <|0 → 15.62 * 10^3, 1 → 15.88 * 10^3, 2 → 16.3 * 10^3, 3 → 16.1 * 10^3, 4 → 15.9 * 10^3, 5 → 15.69 * 10^3|>
   ,
   (* Rabi frequency of single rotations on each qubit *)
   RabiFreq → <|0 → 5, 1 → 5, 2 → 5, 3 → 5, 4 → 5, 5 → 5|>
   ,
   (* Set the noise form of off-resonant Rabi oscillation. This takes RabiFreq information to produce the noise.*)
   OffResonantRabi → True
```

```
,
(* Set the standard depolarising and dephasing passive noise using T1 and T2 *)
StdPassiveNoise → True
,
(* Fidelities of X- and Y- rotations by random benchmarking *)
FidSingleXY → <|0 → 0.9977, 1 → 0.9987, 2 → 0.9996, 3 → 0.9988, 4 → 0.9991, 5 → 0.9989|>
,
(* Error fraction/ratio {depolarising, dephasing} sum is either one or zero *)
EFSingleXY → {0, 1}
,
(*  The rabi Frequency and fidelities of controlled-Z(C[Z]), nearest-neighbors. Keys are the smallest qubit number. This applies to controlled-Ph gates *)
FreqCZ → <|0 → 12.1, 1 → 11.1, 2 → 6.6, 3 → 9.8, 4 → 5.4|>
,
(* Fidelity of controlled-Z; the numbers shown here are obtained from a simple optimisation via bell state fidelity *)
FidCZ → <|0 → 0.9374945614729504`, 1 → 0.9339691831083574`, 2 → 0.9286379436705322`, 3 → 0.9967228426036524`, 4 → 0.9793017377403548`|>
,
(* Fidelity of CROT/Controlled-X rotation *)
FidCRot → 0.9988
,
(* Rabi frequency of CROT, obtained by conditional microwave drive *)
FreqCRot → 5
,
(* Error fraction/ratio {depolarising, dephasing} of controled-Ph(π) or controlled-Z. The error for other θ is scalled from π. *)
EFCZ → {0, 1}
,
(* Crosstalks error (C-Rz[ex])on the passive qubits when applying CZ gates; square matrix with dims nqubit-2 *)
ExchangeRotOn → {{0, 0.023, 0.018 , 0.03, 0.04}, {0.05 , 0, 0.03, 0.03, 0.04}, {0.05, 0.03 , 0, 0.07, 0.042}, {0.038 , 0.03 , 0.031, 0 , 0.25}, {0.033, 0.03 , 0.02, 0.03, 0}}
,
(* Crosstalks error (C-Rz[ex])on the passive qubits when no CZ gates applied; the qubits below indicate the controlled-qubit *)
ExchangeRotOff → <|0 → 0.039, 1 → 0.015 , 2 → 0.03 , 3 → 0.02, 4 → 0.028|>
,
(* Parity readout fidelity/charge readout fidelity between Q0,Q1 or Q5,Q6 *)
FidRead → 0.9997
,
(*Parity readout duration  *)
DurRead → 10
};
```

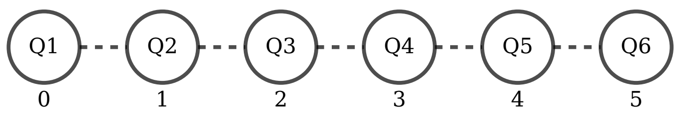
Elementary guide

Device connectivity is nearest-neighbour

In[105]:=

```
With[{nq = OptionValue[SiliconDelft, QubitNum]},
 nodes = Labeled[#, Placed[{#, "Q" <> ToString[# + 1]}, {Below, Center}]] & /@ Range[0, nq - 1];
 Graph[nodes, Table[j ⟷ j + 1, {j, nodes[[All, 1]][[ ;; -2]]}], VertexSize → 0.6,
  VertexStyle → Directive[White, EdgeForm[Thick]], BaseStyle → {11, FontFamily → "Serif"}, ImageSize → Automatic, EdgeStyle → Directive[Black, Thick, Dashed]]
]
```

Out[105]=



Native gates: $Rx_j[\theta]$, $Ry_j[\theta]$, $C_i[X_j]$, $C_i[Ph_j[\theta]]$, Read, Init, $Wait_i[\Delta t]$

Native operations

Initialisation must be done from edge qubits, for example:

$Init_{q1,q2}$, $Init_{q1,q2,q3}$, $Init_{q5,q6}$, $Init_{q4,q5,q6}$

Parity readout only on edge qubits, for example

$MeasP_{q1,q2}$, $MeasP_{q5,q6}$

Single-qubit gates

$Rx_q[\theta]$, $Ry_q[\theta]$, $Rz_q[\theta]$

Two-qubit gates

$C_p[Z_q]$, $C_p[Ph_q]$,

others: doing nothing

$Wait_q$[duration]

Basic operations

Doing nothing, observe the passive noise

The passive noise when no gates are applied.
The noise forms:
1) Cross-talk $C_i[Rz_{i+1}(\Delta t)]$ from input ExchangeRotOff, which is exchange rotation when no C[Z] gate is applied.
2) Standard dephasing from T2 input and depolarising from T1 inputs. We assume the T2* is echoed out to T2

The standard passivenoise (2) can be eliminated by setting **StdPassiveNoise → False**. The Cross-talk (1) can be set off by setting **ExchangeRotOff → False**

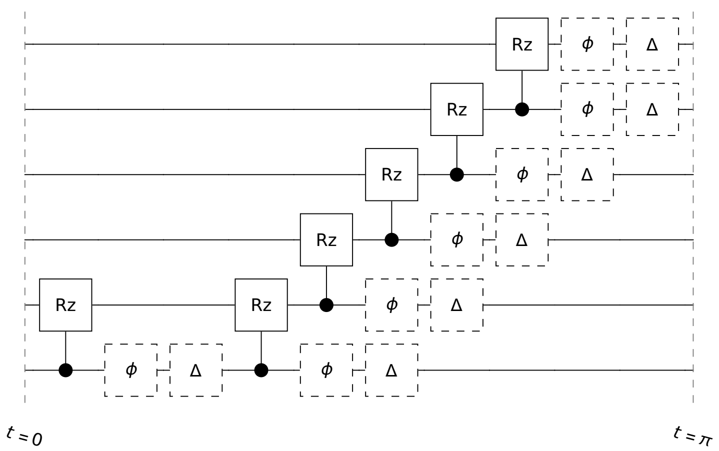
In[106]:=

InsertCircuitNoise[{$Wait_0[\pi]$}, SiliconDelft[]]
DrawCircuit[%]

Out[106]=

{{0, {$C_0[Rz_1[0.039]]$, $Deph_0[0.100502]$, $Depol_0[0.000235582]$}, {$C_0[Rz_1[0.]]$, $Deph_0[0.]$, $Depol_0[0.]$, $C_1[Rz_2[0.015]]$, $Deph_1[0.0691683]$, $Depol_1[0.000235582]$, $C_2[Rz_3[0.03]]$, $Deph_2[0.0376768]$, $Depol_2[0.000235582]$, $C_3[Rz_4[0.02]]$, $Deph_3[0.0404918]$, $Depol_3[0.000235582]$, $C_4[Rz_5[0.028]]$, $Deph_4[0.0339344]$, $Depol_4[0.000235582]$, $Deph_5[0.055502]$, $Depol_5[0.000235582]$}}, {π, {}, {}}}

Out[107]=



Single qubit gates

1) Off-resonant Rabi Oscillation. This takes RabiFreq input and applied when **OffResonantRabi→True**.

2) Standard Dephasing and Depolarising noise. This takes **FidSingleXY** and **EFSingleXY** inputs to estimate the error parameters. Set **EFSingleXY→{0,0}** to set this off.

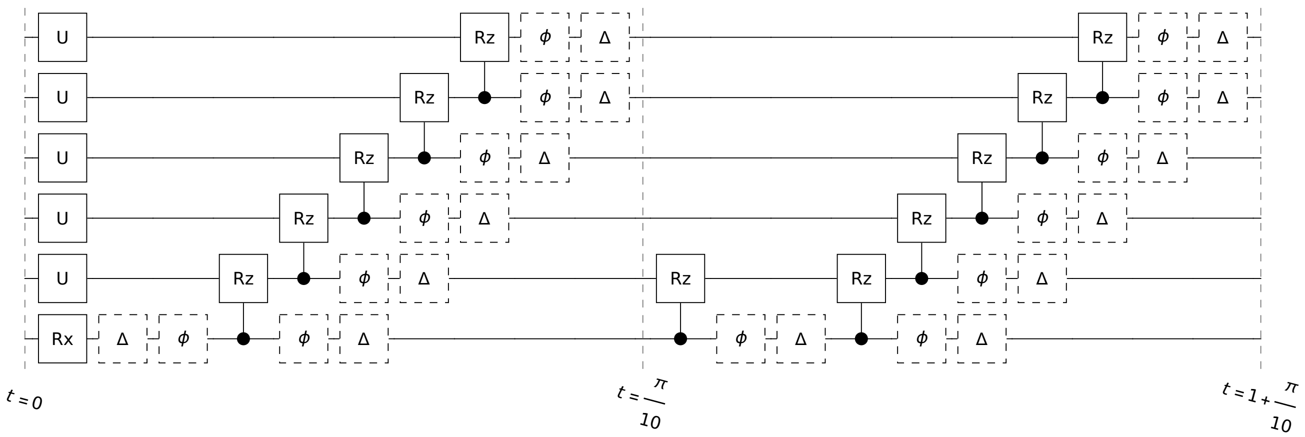
In[108]:=

```
InsertCircuitNoise[CircSiliconDelft[{Rx0[π/2], Wait0[1]}, Parallel → False], SiliconDelft[StdPassiveNoise → True, OffResonantRabi → True]]
DrawCircuit[%]
```

Out[108]=

```
{{0, {Rx0[π/2], Depol0[0.], Deph0[0.001725], U1[{{0.999287 - 0.0377401 i, -1.35127 × 10^-17 - 0.000725771 i}, {-1.35127 × 10^-17 - 0.000725771 i, 0.999287 + 0.0377401 i}}],
   U2[{{0.999896 - 0.0144364 i, -1.45569 × 10^-18 - 0.00010615 i}, {-1.45569 × 10^-18 - 0.00010615 i, 0.999896 + 0.0144364 i}}],
   U3[{{0.999791 - 0.02045 i, 3.05645 × 10^-16 - 0.000213021 i}, {3.05645 × 10^-16 - 0.000213021 i, 0.999791 + 0.02045 i}}],
   U4[{{0.999385 - 0.0350469 i, -9.81184 × 10^-18 - 0.000625837 i}, {-9.81184 × 10^-18 - 0.000625837 i, 0.999385 + 0.0350469 i}}],
   U5[{{0.98769 - 0.139614 i, 0.0705493 - 2.76517 × 10^-16 i}, {0.0705493 - 2.76517 × 10^-16 i, -0.98769 - 0.139614 i}}]},
  {C0[Rz1[0.]], Deph0[0.], Depol0[0.], C1[Rz2[0.0015]], Deph1[0.00738939], Depol1[0.0000235616], C2[Rz3[0.003]], Deph2[0.00390189], Depol2[0.0000235616], C3[Rz4[0.002]],
   Deph3[0.00420479], Depol3[0.0000235616], C4[Rz5[0.0028]], Deph4[0.00350177], Depol4[0.0000235616], Deph5[0.00584866], Depol5[0.0000235616]}},
 {π/10, {C0[Rz1[0.0124141]], Deph0[0.0344686], Depol0[0.0000749963]}, {C0[Rz1[0.]], Deph0[0.], Depol0[0.], C1[Rz2[0.00477465]], Deph1[0.0231439], Depol1[0.0000749963], C2[Rz3[0.0095493]], Deph2[0.0123146],
   Depol2[0.0000749963], C3[Rz4[0.0063662]], Deph3[0.0132618], Depol3[0.0000749963], C4[Rz5[0.00891268]], Deph4[0.0110615], Depol4[0.0000749963], Deph5[0.0183802], Depol5[0.0000749963]}}, {1 + π/10, {}, {}}}
```

Out[109]=



Controlled - Z, Coltrolled-Phase

The noisy forms:

1) Standard 2-qubits dephasing and depolarising noise. This takes information **FidCZ** (fidelities) and **EFCZ** (error fraction). Set it off by **EFCZ→{0,0}**

2) Exchange rotation when a two-qubit gate is on. Takes information **ExchangeRotOn**. Set **ExchangeRotOn → False** to turn it off.

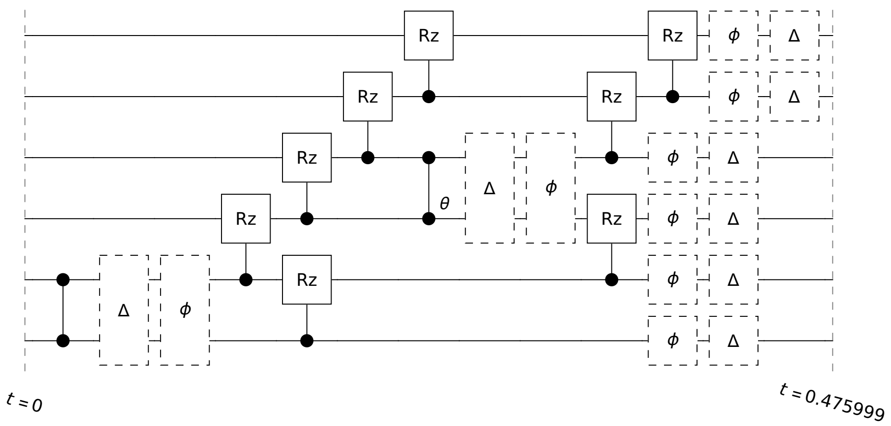
In[110]:=

InsertCircuitNoise[{$C_0[Z_1]$, $C_2[Ph_3[\pi]]$}, SiliconDelft[StdPassiveNoise → True, EFCZ → {0, 0}]]
DrawCircuit[%]

Out[110]=

{{0, {$C_0[Z_1]$, $Depol_{0,1}[0]$, $Deph_{0,1}[0]$, $C_1[Rz_2[0.023]]$, $C_2[Rz_3[0.018]]$, $C_3[Rz_4[0.03]]$, $C_4[Rz_5[0.04]]$, $C_2[Ph_3[\pi]]$, $Depol_{2,3}[0]$, $Deph_{2,3}[0]$, $C_0[Rz_1[0.05]]$, $C_1[Rz_2[0.03]]$, $C_3[Rz_4[0.07]]$, $C_4[Rz_5[0.042]]$}, {$Deph_0[0.00766785]$, $Depol_0[0.0000162271]$, $Deph_1[0.00510089]$, $Depol_1[0.0000162271]$, $Deph_2[0.]$, $Depol_2[0.]$, $Deph_3[0.]$, $Depol_3[0.]$, $Deph_4[0.00529612]$, $Depol_4[0.0000356991]$, $Deph_5[0.00883485]$, $Depol_5[0.0000356991]$}}, {0.475999, {}, {}}}

Out[111]=



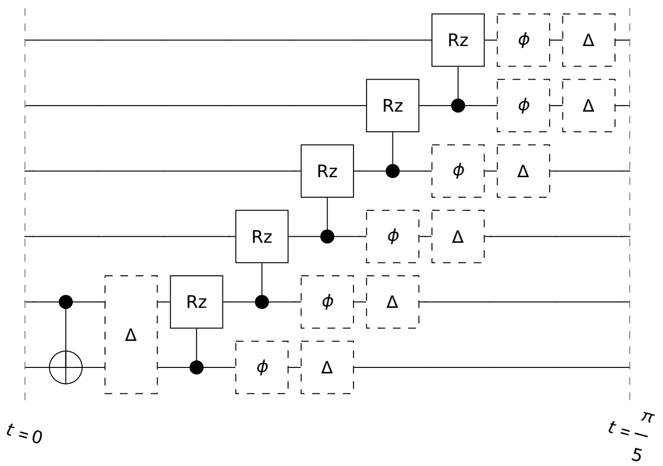
In[112]:=

InsertCircuitNoise[{$CRot_{1,0}$}, SiliconDelft[], ReplaceAliases → False]
DrawCircuit[%]

Out[112]=

{{0, {$C_1[X_0]$, $Depol_{1,0}[0.0012]$}, {$C_0[Rz_1[0.]]$, $Deph_0[0.]$, $Depol_0[0.]$, $C_1[Rz_2[0.]]$, $Deph_1[0.]$, $Depol_1[0.]$, $C_2[Rz_3[0.006]]$, $Deph_2[0.00777334]$, $Depol_2[0.0000471224]$, $C_3[Rz_4[0.004]]$, $Deph_3[0.00837422]$, $Depol_3[0.0000471224]$, $C_4[Rz_5[0.0056]]$, $Deph_4[0.00697901]$, $Depol_4[0.0000471224]$, $Deph_5[0.0116289]$, $Depol_5[0.0000471224]$}}, {$\frac{\pi}{5}$, {}, {}}}

Out[113]=



Readout: parity measurement

Note: see *../supplement/BellsonSiliconDelft/SiDelftReadInit.nb* for further details on this measurement model

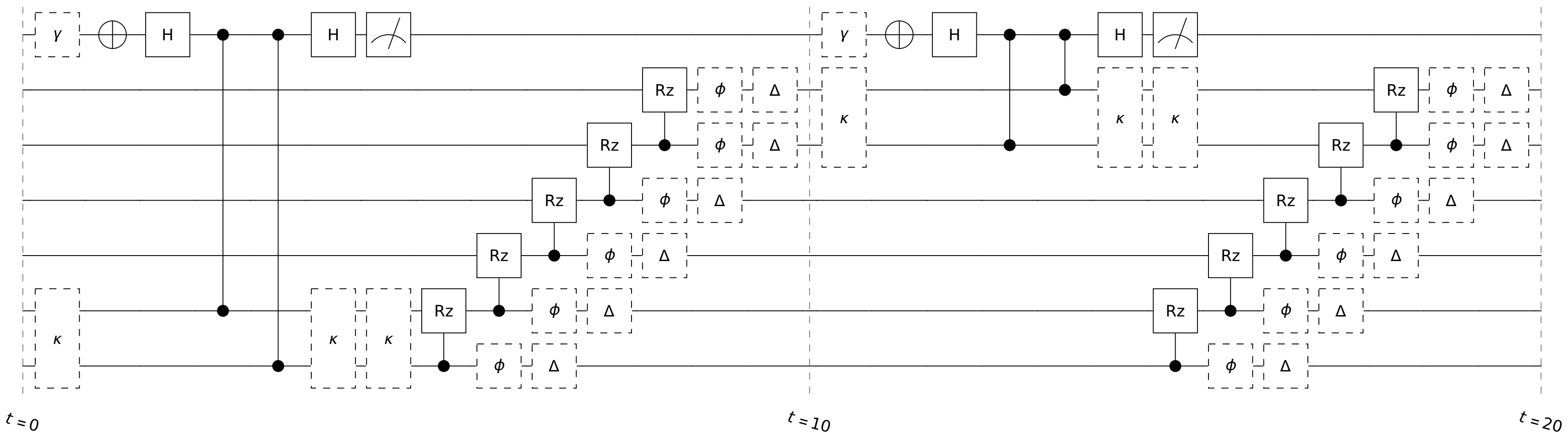
In[114]:=

InsertCircuitNoise[List /@ {$MeasP_{1,0}$, $MeasP_{4,5}$}, SiliconDelft[], ReplaceAliases → True];

In[115]:=

```
DrawCircuit@%
```

Out[115]=



Reproducing results from the paper

Initialisation is obtained by 2 readouts and a partial swap with single qubit errors
The qubits are initialised to state 100 ... 001

Realtime feedback initialisation to $|10\rangle$ and $|100\rangle$

In[116]:=

```
ρ = CreateDensityQureg[7];
```

The initialisation process done in the experiment paper

In[161]:=

```
SetAttributes[readInit, HoldFirst]
readInit[ρ_, q0_, q1_, opt_ : {}] := Module[{m1, m2, dev},
  dev = SiliconDelft[Sequence @@ opt];
  m1 = First@Flatten@ApplyCircuit[ρ, ExtractCircuit@InsertCircuitNoise[{MeasP_{q0,q1}}, dev, ReplaceAliases → True]];
  If[m1 == 1, ApplyCircuit[ρ, ExtractCircuit@InsertCircuitNoise[{Rx_{q0}[π]}, dev, ReplaceAliases → True]]];
  m2 = First@Flatten@ApplyCircuit[ρ, ExtractCircuit@InsertCircuitNoise[{MeasP_{q0,q1}}, SiliconDelft[Sequence @@ opt], ReplaceAliases → True]];
  {m1, m2}
 ]
```

Initialise the qubits to mixed state for a proper test

In[119]:=

```
SetQuregMatrix[ρ, RandomMixState[7]];
readInit[ρ, 0, 1]
Re@PartialTrace[ρ, 2, 3, 4, 5, 6][[2, 2]]
```

Out[120]=

```
{0, 0}
```

Out[121]=

```
0.9997
```

Readout (QND) on middle qubits Q2,Q3 and initialising it at the same time to state $|100\rangle$
Only works if the output the last measurement is 0; repeat the initialisation process otherwise

In[163]:=
```
readInit3[ρ_, q0_, q1_, q2_, opt_ : {}] := Module[{m1, m2, m3, dev},
  dev = SiliconDelft[Sequence @@ opt];
  m1 = First@Flatten@ApplyCircuit[ρ, ExtractCircuit@InsertCircuitNoise[{MeasP_{q0,q1}}, dev, ReplaceAliases → True]];
  If[m1 == 1, ApplyCircuit[ρ, ExtractCircuit@InsertCircuitNoise[{Rx_{q0}[π]}, dev, ReplaceAliases → True]]];
  m2 = First@Flatten@ApplyCircuit[ρ, ExtractCircuit@InsertCircuitNoise[{MeasP_{q0,q1}}, dev, ReplaceAliases → True]];
  m3 = First@Flatten@ApplyCircuit[ρ, ExtractCircuit@InsertCircuitNoise[{CRot_{q2,q1}, MeasP_{q0,q1}}, dev, ReplaceAliases → True]];
  If[m3 == 1, ApplyCircuit[ρ, ExtractCircuit@InsertCircuitNoise[{Rx_{q2}[π]}, dev, ReplaceAliases → True]]];
  {m1, m2, m3}
 ]
```

In[123]:=
```
(* initialise the qubits to mixed state *)
SetQuregMatrix[ρ, RandomMixState[7]];
```

In[124]:=
```
(* Keep doing it until the fidelity is high: Readout repetition in practice *)
```

In[166]:=
```
readInit3[ρ, 0, 1, 2]
Re@PartialTrace[ρ, 3, 4, 5, 6][[2, 2]]
```

Out[166]=
```
{0, 0, 0}
```

Out[167]=
```
0.998879
```

In[168]:=
```
readInit3[ρ, 5, 4, 3]
Re@PartialTrace[ρ, 0, 1, 2, 6][[5, 5]]
```

Out[168]=
```
{0, 0, 0}
```

Out[169]=
```
0.99887
```

Full device initialisation to $|100\,001\rangle$

In[170]:=
```
ψ5 = CreateQureg[7];
ApplyCircuit[InitZeroState@ψ5, {X_0, X_5}];
```

In[172]:=
```
(* initialise the qubits to mixed state *)
SetQuregMatrix[ρ, RandomMixState[7]];
(* repeat the measurement process: ideally all outputs are zeros *)
repeat = 4;

opt = {};
Table[readInit3[ρ, 5, 4, 3, opt], {repeat}]
Table[readInit3[ρ, 0, 1, 2, opt], {repeat}]
CalcFidelity[ρ, ψ5]
```

Out[175]=
```
{{0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}}
```

Out[176]=
```
{{0, 0, 1}, {1, 0, 0}, {0, 0, 0}, {0, 0, 0}}
```

Out[177]=
```
0.979074
```

Obtaining T1: X180-τ- with perfect measurement

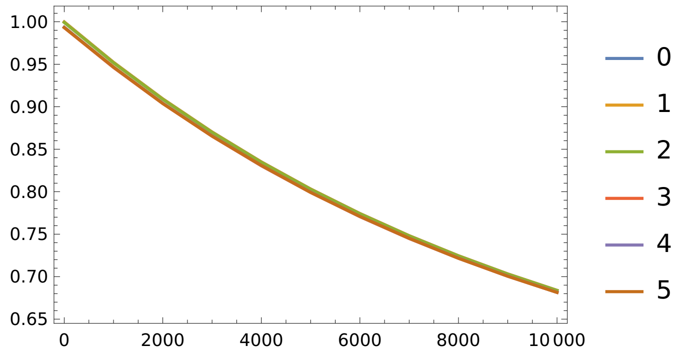
In[178]:=

```
RelaxationExperiment[dev_, qubit_, initrep_ : 4] := Module[{init},
  Table[
   (* initialise the qubits to mixed state *)
   SetQuregMatrix[ρ, RandomMixState[7]];
   Table[readInit3[ρ, 5, 4, 3], {initrep}];
   Table[readInit3[ρ, 0, 1, 2], {initrep}];
   ApplyCircuit[ρ, ExtractCircuit@InsertCircuitNoise[If[MemberQ[{0, 5}, qubit], {Wait_0[t]}, List /@ {Rx_qubit[π], Wait_0[t]}, dev], dev]];
   {t, CalcProbOfOutcome[ρ, qubit, 1]}, {t, 0, 10^4, 1000}]
 ]
```

In[179]:=

```
ListPlot[RelaxationExperiment[SiliconDelft[], #] & /@ Range[0, 5], PlotLegends → Range[0, 5], Joined → True, ImageSize → 300, AxesLabel → {"τ", "p(0)"}, Frame → True]
```

Out[179]=



Obtaining T_2 : X90 - tau - X180 - tau - X90, tau = 1 μs

In[180]:=

```
HahnEchoExperiment[dev_, qubit_, initrep_ : 4] := Module[{},
  Table[
   (* initialise the qubits to mixed state *)
   SetQuregMatrix[ρ, RandomMixState[7]];
   Table[readInit3[ρ, 5, 4, 3], {initrep}];
   Table[readInit3[ρ, 0, 1, 2], {initrep}];
   ApplyCircuit[ρ, ExtractCircuit@InsertCircuitNoise[List /@ {Rx_qubit[π / 2], Wait_0[t / 2], Rx_qubit[π], Wait_0[t / 2], Rx_qubit[π / 2]}, dev]];
   {t, CalcProbOfOutcome[ρ, qubit, If[MemberQ[{5, 0}, qubit], 1, 0]]}, {t, 0, 60, 4}]
 ]
```

In[181]:=

```
OptionValue[SiliconDelft, T2]
```

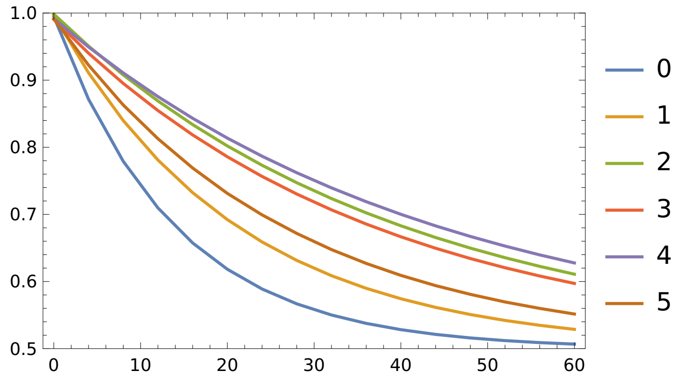
Out[181]=

```
<|0 → 14, 1 → 21.1, 2 → 40.1, 3 → 37.2, 4 → 44.7, 5 → 26.7|>
```

In[182]:=

```
ListPlot[HahnEchoExperiment[SiliconDelft[], #] & /@ Range[0, 5], PlotLegends → Range[0, 5], PlotRange → {0.5, 1}, Joined → True, ImageSize → 300, AxesLabel → {"τ", "p(|+⟩)"}, Frame → True]
```

Out[182]=



Paper supplement: Bell states

In[142]:=

```
chartstyle[label_] :=
  {ImageSize → 200, BarSpacing → 0.1`, ColorFunction → Function[{height}, If[height < 0.1, ColorData["Rainbow"][10 height], ColorData["DeepSeaColors"][(height - 0.9) * 10]]], ChartElementFunction → "Cube",
   ChartStyle → EdgeForm[Thick], PlotTheme → "Business", Ticks → {{{1, "00"}, {2, "01"}, {3, "10"}, {4, "11"}}, {{1, "00"}, {2, "01"}, {3, "10"}, {4, "11"}}, Automatic}, LabelStyle → Directive[Bold, Black],
   Epilog → Inset[Style[label, Thick, 17], ImageScaled[{.2, .7}]], PlotRange → All
  };
```

The CZ gates’ fidelities are unknown. We set it according to the results on the Bell states fidelity.

In[143]:=

```
ψ2 = CreateQureg[2];
ρ2 = CreateDensityQureg[2];
```

In[185]:=

```
concurence[ρ_] := Module[{eigv, ρm, ρms, ρt, nq, pauliy},
  ρm = GetQuregMatrix@ρ;
  nq = IntegerPart@Log2@Length@ρm;
  ρms = MatrixPower[ρm, 1 / 2];
  pauliy = CalcCircuitMatrix[Y# & /@ Range[0, nq - 1]];
  ρt = pauliy.Conjugate[ρm].pauliy;
  eigv = Reverse@Sort[Chop@Eigenvalues[MatrixPower[ρms.ρt.ρms, 1 / 2]]];
  Max[0, eigv[[1]] - Total@eigv[[2 ;;]]]
 ]
```

In[183]:=

```
bellcircρ = <|
   "01" → {Rx0[π / 2], Rx1[-π / 2], C0[Z1], Rx1[π / 2]},
   "12" → {Rx1[π / 2], Rx2[π / 2], C1[Z2], Rx2[π / 2]},
   "23" → {Rx2[π / 2], Rx3[π / 2], C2[Z3], Rx3[π / 2]},
   "34" → {Rx3[π / 2], Rx4[π / 2], C3[Z4], Rx4[π / 2]},
   "45" → {Rx4[π / 2], Rx5[π / 2], C4[Z5], Rx5[π / 2]}|>;
bellcircψ = <|
   "01" → {X0, Rx0[π / 2], Rx1[-π / 2], C0[Z1], Rx1[π / 2]},
   "12" → {Rx0[π / 2], Rx1[π / 2], C0[Z1], Rx1[π / 2]},
   "23" → {Rx0[π / 2], Rx1[π / 2], C0[Z1], Rx1[π / 2]},
   "34" → {Rx0[π / 2], Rx1[π / 2], C0[Z1], Rx1[π / 2]},
   "45" → {X1, Rx0[π / 2], Rx1[π / 2], C0[Z1], Rx1[π / 2]}|>;
```

In[186]:=

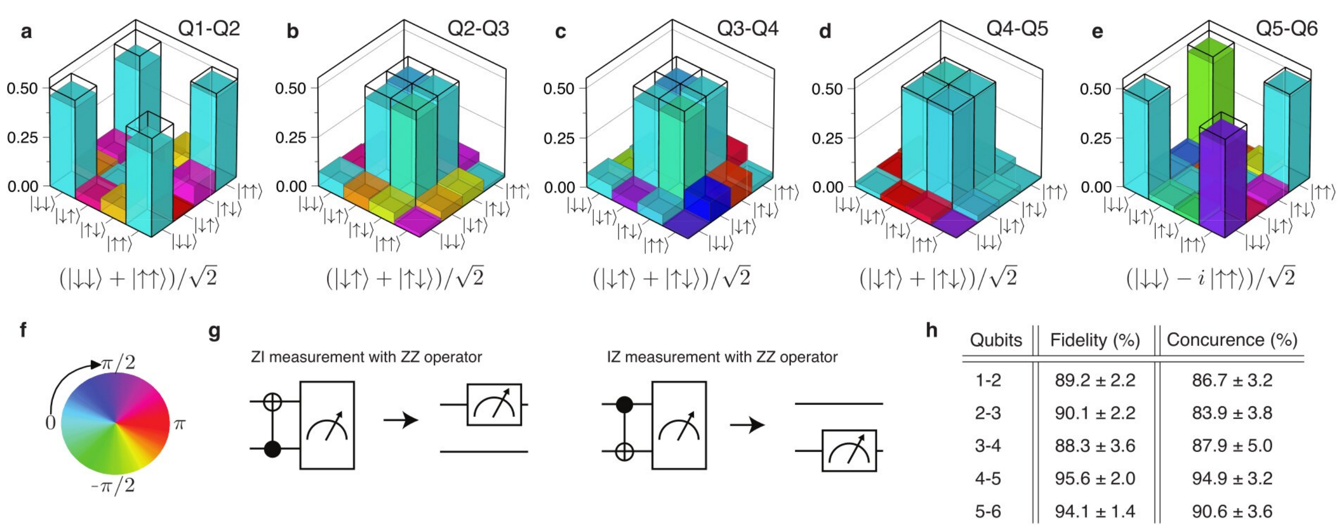
```
bell[code_, opt_, initrep_ : 4] := Module[{qubits, fid, plot, conc, str, out1, out2},
  qubits = ToExpression@StringSplit[code, ""];

  SetQuregMatrix[ρ, IdentityMatrix[2^7]];
  out1 = Table[readInit3[ρ, 5, 4, 3, opt], {initrep}];
  out2 = Table[readInit3[ρ, 0, 1, 2, opt], {initrep}];

  ApplyCircuit[ρ, ExtractCircuit@InsertCircuitNoise[Serialize@{bellcircρ[code]}, SiliconDelft[Sequence @@ opt]]];
  SetQuregMatrix[ρ2, PartialTrace[ρ, Sequence @@ Complement[Range[0, 5], qubits], 6]];
  conc = concurence[ρ2] * 100;
  ApplyCircuit[InitZeroState@ψ2, bellcircψ[code]];
  fid = CalcFidelity[ρ2, ψ2] * 100;
  str = "Q" <> ToString[1 + qubits[[1]]] <> "-Q" <> ToString[1 + qubits[[2]]];
  plot = PlotDensityMatrix[ρ2, ψ2, Sequence @@ chartstyle[str]];
  {str, plot, fid, conc}
 ]
```

Reference from the experiment

In[ ]:=



Qubits	Fidelity (%)	Concurence (%)
1-2	89.2 ± 2.2	86.7 ± 3.2
2-3	90.1 ± 2.2	83.9 ± 3.8
3-4	88.3 ± 3.6	87.9 ± 5.0
4-5	95.6 ± 2.0	94.9 ± 3.2
5-6	94.1 ± 1.4	90.6 ± 3.6

The simulation

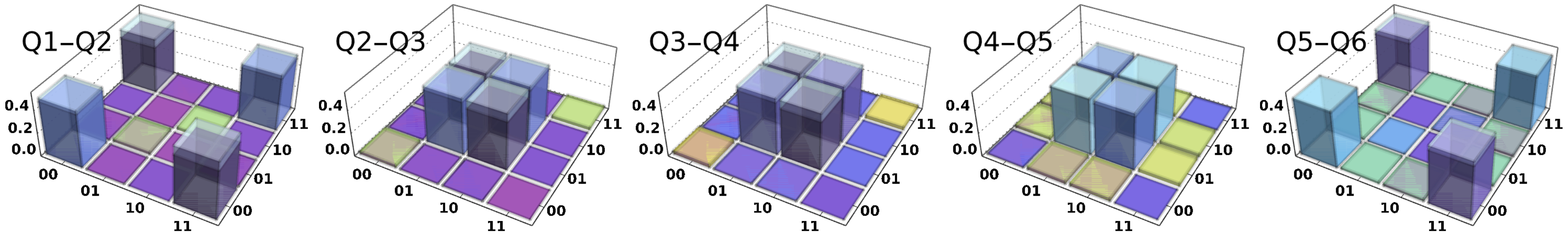
In[199]:=

```
plots = Transpose[bell[#, {}, 4] & /@ ReplaceList[Sort@Range[0, 5], {p___, a_, b_, q___} :→ StringRiffle[{a, b}, ""]]];
TableForm[Transpose@{DecimalForm[#, 4] & /@ plots[[3]], DecimalForm[#, 4] & /@ plots[[4]]}, TableHeadings → {plots[[1]], {"Fidelity", "Concurence"}}]
Row@plots[[2]]
```

Out[200]//TableForm=

	Fidelity	Concurence
Q1–Q2	89.4	79.75
Q2–Q3	90.18	80.42
Q3–Q4	88.69	79.06
Q4–Q5	95.94	94.43
Q5–Q6	94.3	90.65

Out[201]=



In[152]:=

```
(*Table[
  Export[plots[[1, i]] <> ".pdf", plots[[2, i]]],
  {i, Length@plots[[1]]}]*)
```

In[202]:=

```
(* average fidelity of CZ gates *)
Round[#, 0.01] & /@ plots[[3]]
Mean@%
```

Out[202]=

{89.4, 90.18, 88.69, 95.94, 94.3}

Out[203]=

91.702