QuTiP lecture: simulation of a two-qubit gate using a resonator as coupler

Author: J.R. Johansson, robert@riken.jp

http://dml.riken.jp/~rob/

Latest version of this ipython notebook lecture is available at: http://github.com/jrjohansson/qutip-lectures

In [1]:
```
%pylab inline
```

```
Welcome to pylab, a matplotlib-based Python environment [backend: module://IPython.zmq.pylab.backend_inline].
For more information, type 'help(pylab)'.
```

In [2]:
```
from qutip import *
```

Parameters

In [3]:
```
N = 10

wc = 5.0 * 2 * pi
w1 = 3.0 * 2 * pi
w2 = 2.0 * 2 * pi

g1 = 0.01 * 2 * pi
g2 = 0.0125 * 2 * pi

tlist = linspace(0, 100, 500)

width = 0.5

# resonant SQRT iSWAP gate
T0_1 = 20
T_gate_1 = (1*pi)/(4 * g1)

# resonant iSWAP gate
T0_2 = 60
T_gate_2 = (2*pi)/(4 * g2)
```

Operators, Hamiltonian and initial state

In [4]:
```
# cavity operators
a = tensor(destroy(N), qeye(2), qeye(2))
n = a.dag() * a

# operators for qubit 1
sm1 = tensor(qeye(N), destroy(2), qeye(2))
sz1 = tensor(qeye(N), sigmaz(), qeye(2))
n1 = sm1.dag() * sm1

# oeprators for qubit 2
sm2 = tensor(qeye(N), qeye(2), destroy(2))
sz2 = tensor(qeye(N), qeye(2), sigmaz())
n2 = sm2.dag() * sm2
```

In [5]:
```
# Hamiltonian using QuTiP
Hc = a.dag() * a
H1 = - 0.5 * sz1
H2 = - 0.5 * sz2
Hc1 = g1 * (a.dag() * sm1 + a * sm1.dag())
Hc2 = g2 * (a.dag() * sm2 + a * sm2.dag())

H = wc * Hc + w1 * H1 + w2 * H2 + Hc1 + Hc2
```

In [6]:
```
H
```

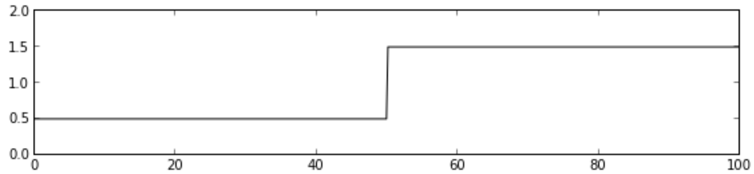
Out [6]: Quantum object: dims = [[10, 2, 2], [10, 2, 2]], shape = [40, 40], type = oper, isHerm = True

$$\left(\begin{array}{ccccccccc}
-15.7079632679 & 0.0 & 0.0 & 0.0 & 0.0 & \cdots & 0.0 & 0.0 & 0.0\\
0.0 & -3.14159265359 & 0.0 & 0.0 & 0.0785398163397 & \cdots & 0.0 & 0.0 & 0.0\\
0.0 & 0.0 & 3.14159265359 & 0.0 & 0.0628318530718 & \cdots & 0.0 & 0.0 & 0.0\\
0.0 & 0.0 & 0.0 & 15.7079632679 & 0.0 & \cdots & 0.0 & 0.0 & 0.0\\
0.0 & 0.0785398163397 & 0.0628318530718 & 0.0 & 15.7079632679 & \cdots & 0.0 & 0.0 & 0.0\\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots\\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & \cdots & 267.035375555 & 0.0 & 0.1884955\\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & \cdots & 0.0 & 267.035375555 & 0.0\\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & \cdots & 0.188495559215 & 0.0 & 279.6017\\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & \cdots & 0.235619449019 & 0.0 & 0.0\\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & \cdots & 0.0 & 0.0 & 0.0\\
\end{array}\right.$$

In [7]:
```python
# initial state: start with one of the qubits in its excited state
psi0 = tensor(basis(N,0),basis(2,1),basis(2,0))
```

Ideal two-qubit iSWAP gate

In [8]:
```python
def step_t(w1, w2, t0, width, t):
    """
    Step function that goes from w1 to w2 at time t0
    as a function of t.
    """
    return w1 + (w2 - w1) * (t > t0)


fig, axes = subplots(1, 1, figsize=(8,2))
axes.plot(tlist, [step_t(0.5, 1.5, 50, 0.0, t) for t in tlist], 'k')
axes.set_ylim(0, 2)
fig.tight_layout()
```



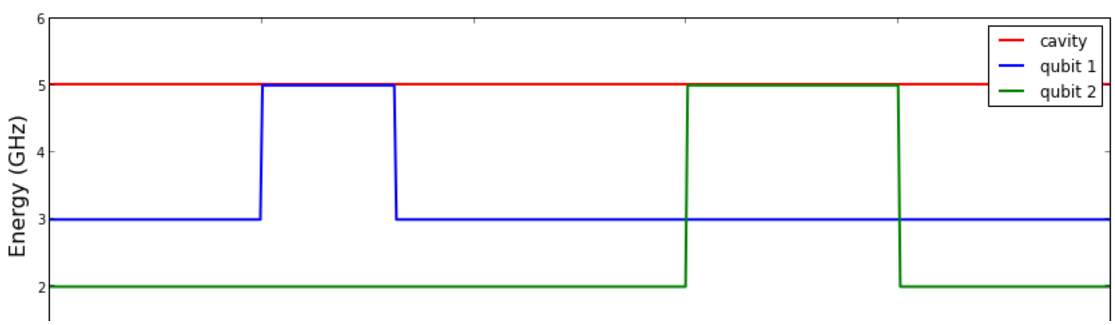
In [9]:
```python
def wc_t(t, args=None):
    return wc

def w1_t(t, args=None):
    return w1 + step_t(0.0, wc-w1, T0_1, width, t) - step_t(0.0, wc-w1, T0_1+T_gate_1, width, t)

def w2_t(t, args=None):
    return w2 + step_t(0.0, wc-w2, T0_2, width, t) - step_t(0.0, wc-w2, T0_2+T_gate_2, width, t)


H_t = [[Hc, wc_t], [H1, w1_t], [H2, w2_t], Hc1+Hc2]
```

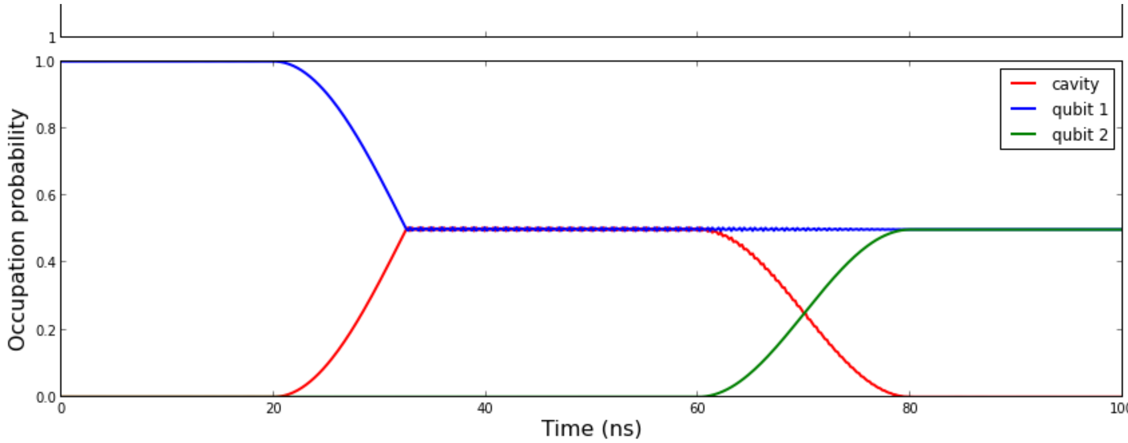
Evolve the system

In [10]:
```python
res = mesolve(H_t, psi0, tlist, [], [])
```

Plot the results

In [11]:
```python
fig, axes = subplots(2, 1, sharex=True, figsize=(12,8))

axes[0].plot(tlist, array(map(wc_t, tlist)) / (2*pi), 'r', linewidth=2, label="cavity")
axes[0].plot(tlist, array(map(w1_t, tlist)) / (2*pi), 'b', linewidth=2, label="qubit 1")
axes[0].plot(tlist, array(map(w2_t, tlist)) / (2*pi), 'g', linewidth=2, label="qubit 2")
axes[0].set_ylim(1, 6)
axes[0].set_ylabel("Energy (GHz)", fontsize=16)
axes[0].legend()

axes[1].plot(tlist, real(expect(n, res.states)), 'r', linewidth=2, label="cavity")
axes[1].plot(tlist, real(expect(n1, res.states)), 'b', linewidth=2, label="qubit 1")
axes[1].plot(tlist, real(expect(n2, res.states)), 'g', linewidth=2, label="qubit 2")
axes[1].set_ylim(0, 1)

axes[1].set_xlabel("Time (ns)", fontsize=16)
axes[1].set_ylabel("Occupation probability", fontsize=16)
axes[1].legend()

fig.tight_layout()
```





Inspect the final state

In [12]:
```
# extract the final state from the result of the simulation
rho_final = res.states[-1]
```

In [13]:
```
# trace out the resonator mode and print the two-qubit density matrix
rho_qubits = ptrace(rho_final, [1,2])
rho_qubits
```

Out [13]: Quantum object: dims = [[2, 2], [2, 2]], shape = [4, 4], type = oper, isHerm = True

$$\begin{pmatrix} 6.15572171515e-05 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.498709211577 & (-0.498492013055+0.0198028691015j) & 0.0 \\ 0.0 & (-0.498492013055-0.0198028691015j) & 0.499061246366 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

In [14]:
```
# compare to the ideal result of the sqrtiswap gate (plus phase correction) for the current initial state
rho_qubits_ideal = ket2dm(tensor(phasegate(0), phasegate(-pi/2)) * sqrtiswap() * tensor(basis(2,1), basis(2,0)))
rho_qubits_ideal
```

Out [14]: Quantum object: dims = [[2, 2], [2, 2]], shape = [4, 4], type = oper, isHerm = True

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.5 & -0.5 & 0.0 \\ 0.0 & -0.5 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

Fidelity and concurrence

In [15]:
```
fidelity(rho_qubits, rho_qubits_ideal)
```

Out [15]: 0.9986877782762704

In [16]:
```
concurrence(rho_qubits)
```

Out [16]: 0.99777039043983007

Dissipative two-qubit iSWAP gate

Define collapse operators that describe dissipation

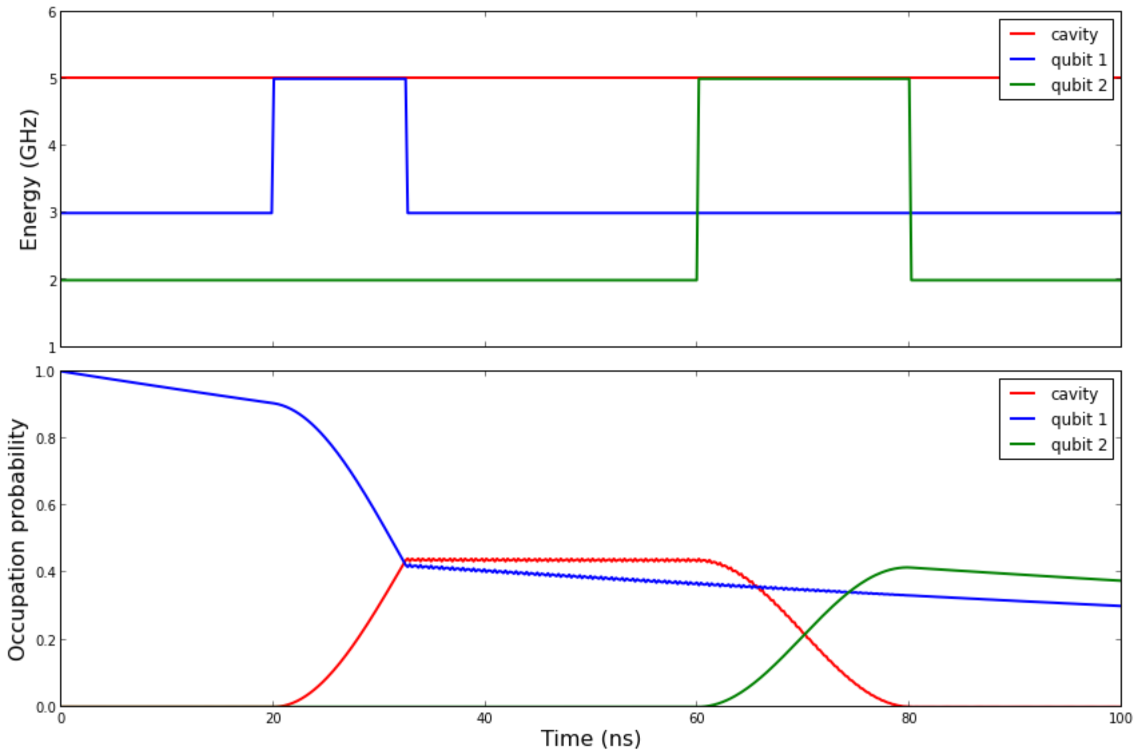
In [17]:
```
kappa = 0.0001
gamma1 = 0.005
gamma2 = 0.005

c_ops = [sqrt(kappa) * a, sqrt(gamma1) * sm1, sqrt(gamma2) * sm2]
```

Evolve the system

In [18]:
```
res = mesolve(H_t, psi0, tlist, c_ops, [])
```

Plot the results

In [19]:
```
fig, axes = subplots(2, 1, sharex=True, figsize=(12,8))

axes[0].plot(tlist, array(map(wc_t, tlist)) / (2*pi), 'r', linewidth=2, label="cavity")
axes[0].plot(tlist, array(map(w1_t, tlist)) / (2*pi), 'b', linewidth=2, label="qubit 1")
axes[0].plot(tlist, array(map(w2_t, tlist)) / (2*pi), 'g', linewidth=2, label="qubit 2")
axes[0].set_ylim(1, 6)
axes[0].set_ylabel("Energy (GHz)", fontsize=16)
axes[0].legend()

axes[1].plot(tlist, real(expect(n, res.states)), 'r', linewidth=2, label="cavity")
axes[1].plot(tlist, real(expect(n1, res.states)), 'b', linewidth=2, label="qubit 1")
axes[1].plot(tlist, real(expect(n2, res.states)), 'g', linewidth=2, label="qubit 2")
axes[1].set_ylim(0, 1)

axes[1].set_xlabel("Time (ns)", fontsize=16)
axes[1].set_ylabel("Occupation probability", fontsize=16)
axes[1].legend()

fig.tight_layout()
```



Fidelity and concurrence

In [20]:
```
rho_final = res.states[-1]
rho_qubits = ptrace(rho_final, [1,2])
```

In [21]:
```
fidelity(rho_qubits, rho_qubits_ideal)
```

Out [21]: 0.8235885404927015

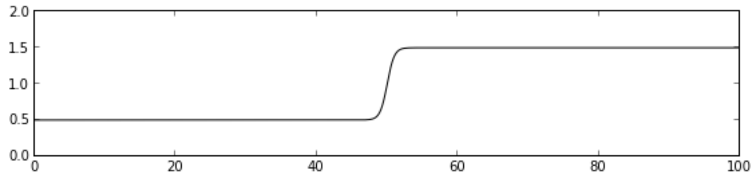
In [22]:
```
concurrence(rho_qubits)
```

Out [22]: 0.67237860360914092

Two-qubit iSWAP gate: Finite pulse rise time

In [23]:
```
def step_t(w1, w2, t0, width, t):
    """
    Step function that goes from w1 to w2 at time t0
    as a function of t, with finite rise time defined
    by the parameter width.
    """
    return w1 + (w2 - w1) / (1 + exp(-(t-t0)/width))
```

```
fig, axes = subplots(1, 1, figsize=(8,2))
axes.plot(tlist, [step_t(0.5, 1.5, 50, width, t) for t in tlist], 'k')
axes.set_ylim(0, 2)
fig.tight_layout()
```

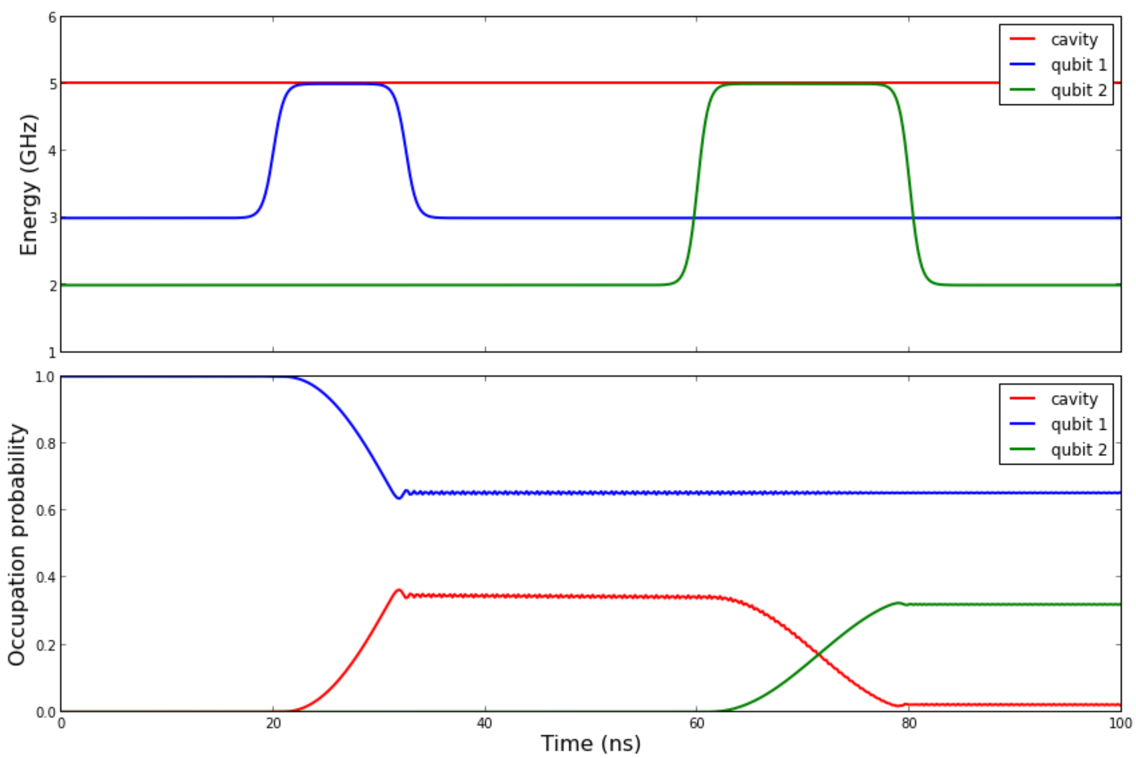


Evolve the system

In [24]:
```
res = mesolve(H_t, psi0, tlist, [], [])
```

Plot the results

In [25]:
```
fig, axes = subplots(2, 1, sharex=True, figsize=(12,8))

axes[0].plot(tlist, array(map(wc_t, tlist)) / (2*pi), 'r', linewidth=2, label="cavity")
axes[0].plot(tlist, array(map(w1_t, tlist)) / (2*pi), 'b', linewidth=2, label="qubit 1")
axes[0].plot(tlist, array(map(w2_t, tlist)) / (2*pi), 'g', linewidth=2, label="qubit 2")
axes[0].set_ylim(1, 6)
axes[0].set_ylabel("Energy (GHz)", fontsize=16)
axes[0].legend()

axes[1].plot(tlist, real(expect(n, res.states)), 'r', linewidth=2, label="cavity")
axes[1].plot(tlist, real(expect(n1, res.states)), 'b', linewidth=2, label="qubit 1")
axes[1].plot(tlist, real(expect(n2, res.states)), 'g', linewidth=2, label="qubit 2")
axes[1].set_ylim(0, 1)

axes[1].set_xlabel("Time (ns)", fontsize=16)
axes[1].set_ylabel("Occupation probability", fontsize=16)
axes[1].legend()

fig.tight_layout()
```



Fidelity and concurrence

In [26]:
```
rho_final = res.states[-1]
rho_qubits = ptrace(rho_final, [1,2])
```

In [27]:
```
fidelity(rho_qubits, rho_qubits_ideal)
```

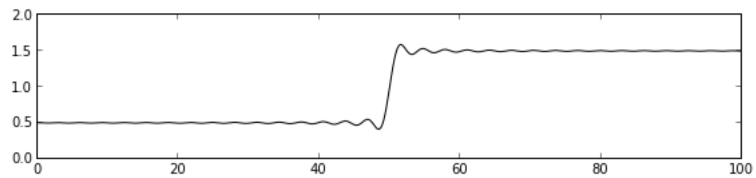
Out [27]: 0.970184996286606

In [28]:
```
concurrence(rho_qubits)
```

Out [28]: 0.91473062460510268

Two-qubit iSWAP gate: Finite rise time with overshoot

In [29]:
```
from scipy.special import sici

def step_t(w1, w2, t0, width, t):
    """
    Step function that goes from w1 to w2 at time t0
    as a function of t, with finite rise time and
    and overshoot defined by the parameter width.
    """

    return w1 + (w2-w1) * (0.5 + sici((t-t0)/width)[0]/(pi))


fig, axes = subplots(1, 1, figsize=(8,2))
axes.plot(tlist, [step_t(0.5, 1.5, 50, width, t) for t in tlist], 'k')
axes.set_ylim(0, 2)
fig.tight_layout()
```

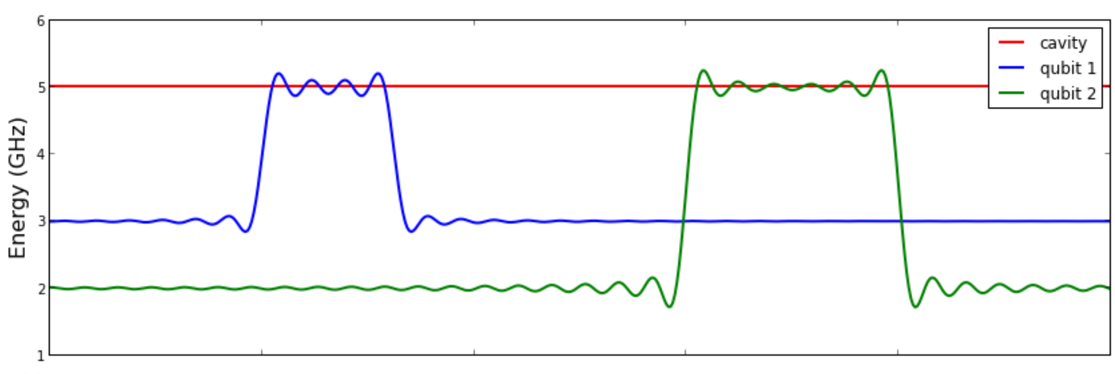


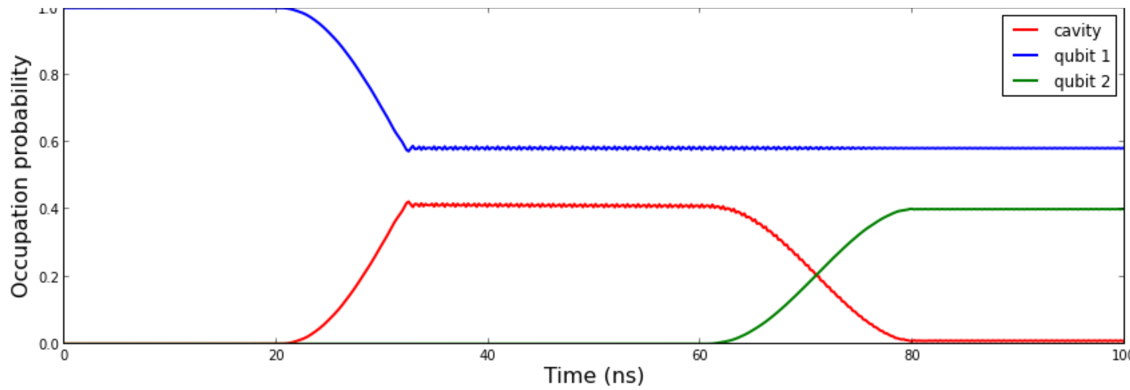
Evolve the system

In [30]:
```
res = mesolve(H_t, psi0, tlist, [], [])
```

Plot the results

In [31]:
```
fig, axes = subplots(2, 1, sharex=True, figsize=(12,8))

axes[0].plot(tlist, array(map(wc_t, tlist)) / (2*pi), 'r', linewidth=2, label="cavity")
axes[0].plot(tlist, array(map(w1_t, tlist)) / (2*pi), 'b', linewidth=2, label="qubit 1")
axes[0].plot(tlist, array(map(w2_t, tlist)) / (2*pi), 'g', linewidth=2, label="qubit 2")
axes[0].set_ylim(1, 6)
axes[0].set_ylabel("Energy (GHz)", fontsize=16)
axes[0].legend()

axes[1].plot(tlist, real(expect(n, res.states)), 'r', linewidth=2, label="cavity")
axes[1].plot(tlist, real(expect(n1, res.states)), 'b', linewidth=2, label="qubit 1")
axes[1].plot(tlist, real(expect(n2, res.states)), 'g', linewidth=2, label="qubit 2")
axes[1].set_ylim(0, 1)

axes[1].set_xlabel("Time (ns)", fontsize=16)
axes[1].set_ylabel("Occupation probability", fontsize=16)
axes[1].legend()

fig.tight_layout()
```





Fidelity and concurrence

```
In [32]: rho_final = res.states[-1]
         rho_qubits = ptrace(rho_final, [1,2])
```

```
In [33]: fidelity(rho_qubits, rho_qubits_ideal)
```

```
Out [33]: 0.9858234626300226
```

```
In [34]: concurrence(rho_qubits)
```

```
Out [34]: 0.96641065049070052
```

Two-qubit iSWAP gate: Finite pulse rise time and dissipation

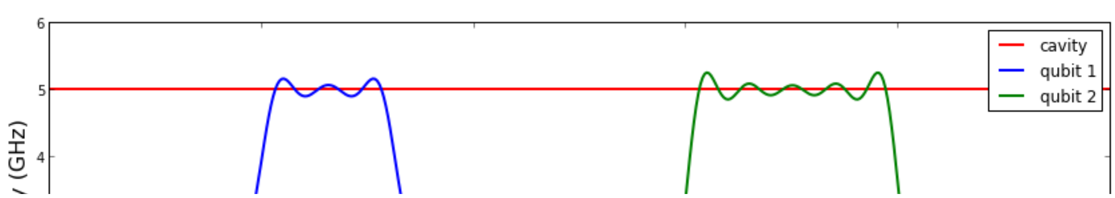
```
In [35]: # increase the pulse rise time a bit
         width = 0.6

         # high-Q resonator but dissipative qubits
         kappa  = 0.00001
         gamma1 = 0.005
         gamma2 = 0.005

         c_ops = [sqrt(kappa) * a, sqrt(gamma1) * sm1, sqrt(gamma2) * sm2]
```

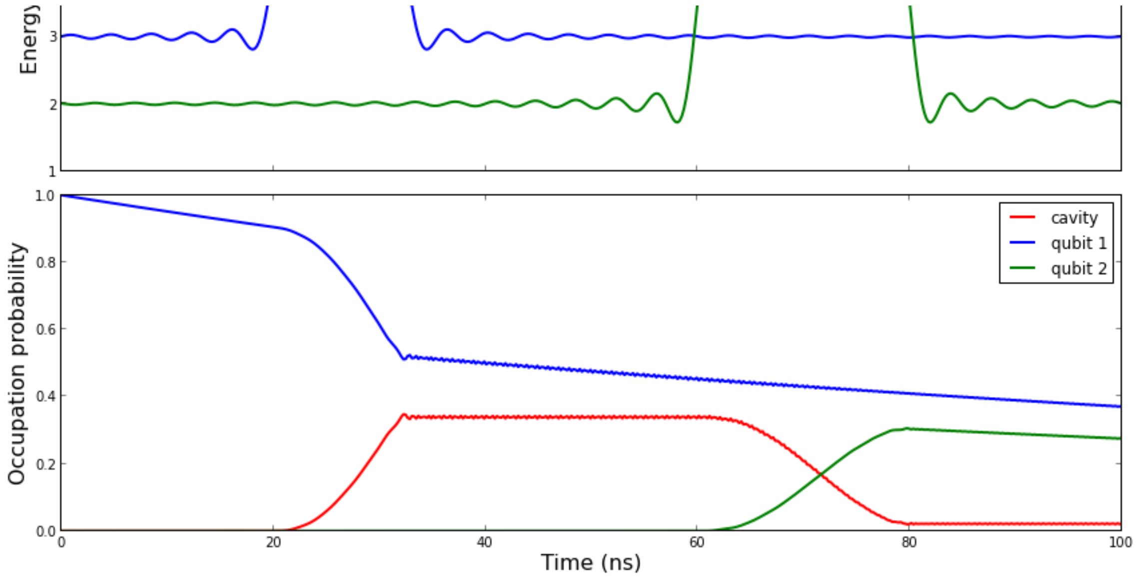
Evolve the system

```
In [36]: res = mesolve(H_t, psi0, tlist, c_ops, [])
```

Plot results

```
In [37]: fig, axes = subplots(2, 1, sharex=True, figsize=(12,8))

         axes[0].plot(tlist, array(map(wc_t, tlist)) / (2*pi), 'r', linewidth=2, label="cavity")
         axes[0].plot(tlist, array(map(w1_t, tlist)) / (2*pi), 'b', linewidth=2, label="qubit 1")
         axes[0].plot(tlist, array(map(w2_t, tlist)) / (2*pi), 'g', linewidth=2, label="qubit 2")
         axes[0].set_ylim(1, 6)
         axes[0].set_ylabel("Energy (GHz)", fontsize=16)
         axes[0].legend()

         axes[1].plot(tlist, real(expect(n, res.states)), 'r', linewidth=2, label="cavity")
         axes[1].plot(tlist, real(expect(n1, res.states)), 'b', linewidth=2, label="qubit 1")
         axes[1].plot(tlist, real(expect(n2, res.states)), 'g', linewidth=2, label="qubit 2")
         axes[1].set_ylim(0, 1)

         axes[1].set_xlabel("Time (ns)", fontsize=16)
         axes[1].set_ylabel("Occupation probability", fontsize=16)
         axes[1].legend()

         fig.tight_layout()
```





Fidelity and concurrence

In [38]:
```
rho_final = res.states[-1]
rho_qubits = ptrace(rho_final, [1,2])
```

In [39]:
```
fidelity(rho_qubits, rho_qubits_ideal)
```

Out [39]: 0.7943108372320334

In [40]:
```
concurrence(rho_qubits)
```

Out [40]: 0.62615691287517516

Two-qubit iSWAP gate: Using tunable resonator and fixed-frequency qubits

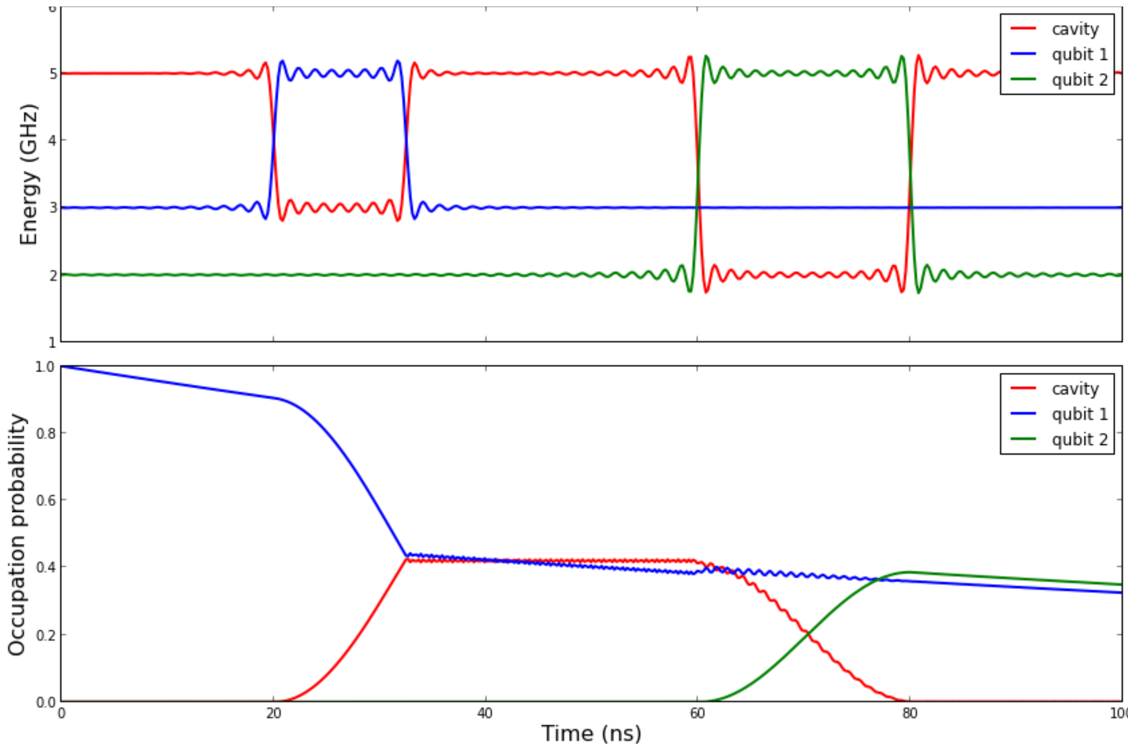
In [41]:
```
# reduce the rise time
width = 0.25

def wc_t(t, args=None):
    return wc - step_t(0.0, wc-w1, T0_1, width, t) + step_t(0.0, wc-w1, T0_1+T_gate_1, width, t) \
              - step_t(0.0, wc-w2, T0_2, width, t) + step_t(0.0, wc-w2, T0_2+T_gate_2, width, t)

H_t = [[Hc, wc_t], H1 * w1 + H2 * w2 + Hc1+Hc2]
```

Evolve the system

In [42]:
```
res = mesolve(H_t, psi0, tlist, c_ops, [])
```

Plot the results

In [43]:
```
fig, axes = subplots(2, 1, sharex=True, figsize=(12,8))

axes[0].plot(tlist, array(map(wc_t, tlist)) / (2*pi), 'r', linewidth=2, label="cavity")
axes[0].plot(tlist, array(map(w1_t, tlist)) / (2*pi), 'b', linewidth=2, label="qubit 1")
axes[0].plot(tlist, array(map(w2_t, tlist)) / (2*pi), 'g', linewidth=2, label="qubit 2")
axes[0].set_ylim(1, 6)
axes[0].set_ylabel("Energy (GHz)", fontsize=16)
axes[0].legend()

axes[1].plot(tlist, real(expect(n, res.states)), 'r', linewidth=2, label="cavity")
axes[1].plot(tlist, real(expect(n1, res.states)), 'b', linewidth=2, label="qubit 1")
axes[1].plot(tlist, real(expect(n2, res.states)), 'g', linewidth=2, label="qubit 2")
axes[1].set_ylim(0, 1)

axes[1].set_xlabel("Time (ns)", fontsize=16)
axes[1].set_ylabel("Occupation probability", fontsize=16)
axes[1].legend()

fig.tight_layout()
```



Fidelity and concurrence

In [44]:
```
rho_final = res.states[-1]
rho_qubits = ptrace(rho_final, [1,2])
```

In [45]:
```
fidelity(rho_qubits, rho_qubits_ideal)
```

Out [45]: 0.8209803465743701

In [46]:
```
concurrence(rho_qubits)
```

Out [46]: 0.67365901098917069