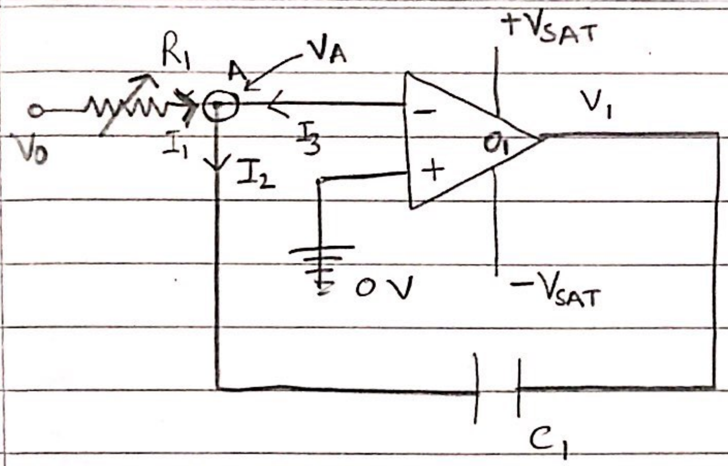
Q1 (a)



According to KCL,

$$+I_1 - I_2 + I_3 = 0$$

$\Rightarrow I_1 = I_2$ $\quad$ ($\because$ $I_3 = 0$ by Golden Rules)

$$\Rightarrow \frac{V_0 - V_A}{R_1} = C\frac{d(V_A - V_1)}{dt}$$

$\because$ this is an inverted op amp,
$\therefore$ by Golden Rules, $V_A = 0V$.

$$\Rightarrow \frac{V_0}{R_1} = \frac{C_1 d(-V_1)}{dt}$$

$$\Rightarrow \frac{V_0}{R_1} = -C_1\frac{dV_1}{dt}$$

$$\Rightarrow dV_1 = -\frac{V_0}{R_1C_1}\, dt$$

Integrating both sides from 0 to t,

$$\Rightarrow V_1 = -\int_0^t \frac{V_0}{R_1C_1}\, dt + \text{constant}$$

The constant is the pre existing output voltage of the integrator at $t = 0$, let's call it $V_{initial}$.

$$\therefore V_1 = -\int_0^t \frac{V_0}{R_1C_1}\, dt + V_{initial} \quad \# \text{ Ans}$$

Date ___/___/___

Page ___

Q1(b) <u>For T_1</u>

$$\text{Slope} = \frac{\text{Rise}}{\text{Run}}$$

$$\Rightarrow \frac{2V_{th}}{T_1} = \frac{V_{SAT}}{R_1 C_1}$$

$$\Rightarrow T_1 = \frac{2V_{th} R_1 C_1}{V_{SAT}}$$

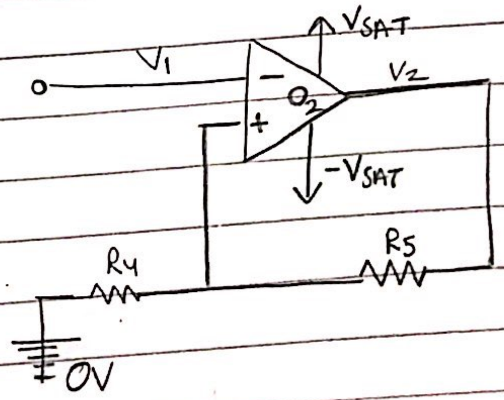
$\because$ $V_{th} = V_{sat}$ (see graph)

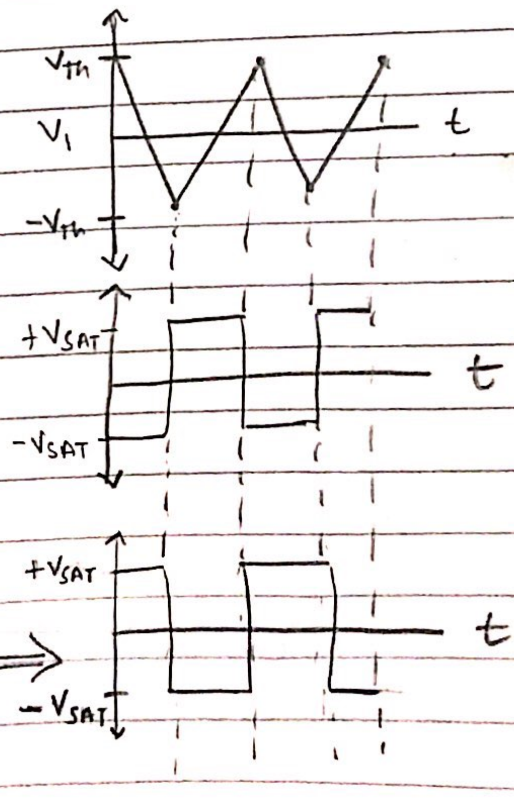
$\therefore$ $T_1 = 2R_1C_1$

$\therefore$ $T_2 = 2R_1C_1$ ($\because$ independent of other factors)

Scanned by CamScanner

Q1 (c)





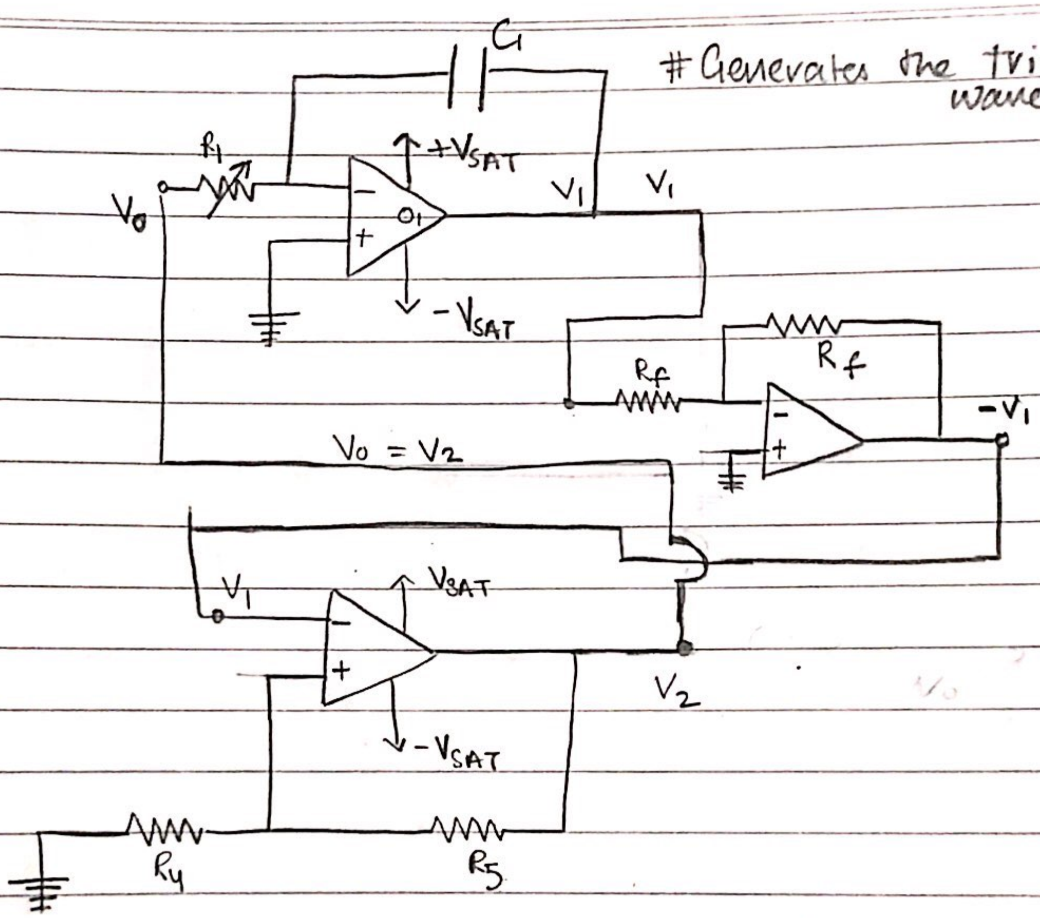
#using V_1

#using $(-V_1)$

$$\pm V_{SAT} = \frac{R_4 + R_5}{R_4} (\pm V_{th})$$

Matches $\Longrightarrow$

Q1(d)



Generates the triangular waveform

Date ___/___/___

Page ________

Q1 (e) $C_1 = 0.01\ \mu F = 1 \times 10^{-8}\ F$

$R_4 = 10\ k\Omega = 10000\ \Omega$

$\pm V_{SAT} = \pm 10\ V$

$\pm V_{Th} = \pm 5V$

$\nu = 1\ kHz = 1000\ Hz$

$\Rightarrow T = \dfrac{1}{1000}\ s$ — $T_1 + T_2 = T$?

FROM PART (b),

$$T_1 = \frac{2V_{Th}}{V_{SAT}} R_1 C_1$$

$$\Rightarrow \frac{1}{1000} = \frac{2 \times 5}{\cancel{10}} \times R_1 \times 1 \times 10^{-8}$$

$$\Rightarrow R_1 \times 10^{3-8} = 1$$

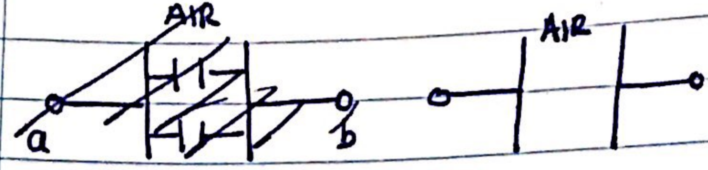
$$\Rightarrow R_1 \times \frac{1}{10^5} = 1$$

$$\Rightarrow R_1 = 10^5 = 100{,}000 = \underline{\underline{100\ k\Omega}}$$

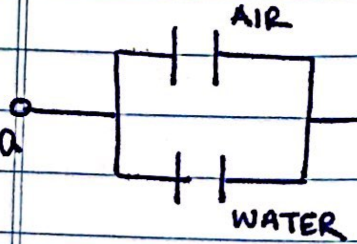
$T_2 =$?

Q1

Q2 (a)



$$C_{empty} = \epsilon_0 \frac{w h_{tot}}{w} = \epsilon_0 h_{tot}$$



$$C_{tank} = C_{AIR} + C_{WATER}$$

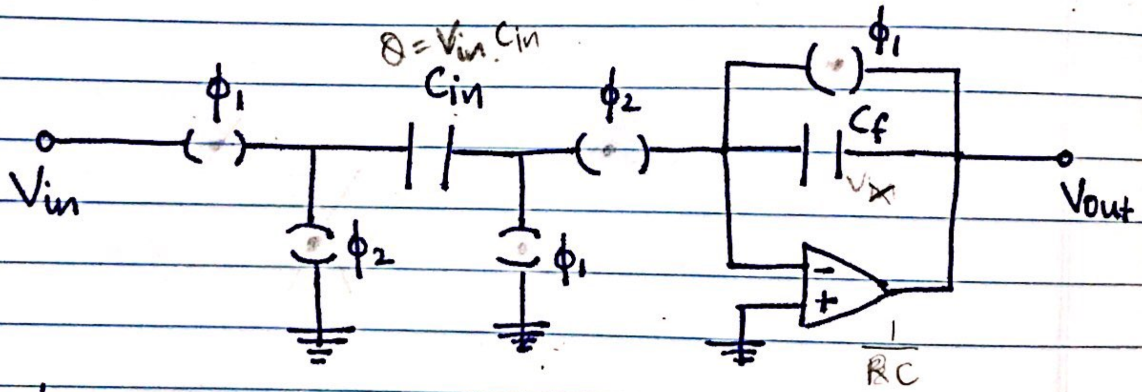
$$= \epsilon_0 \frac{w(h_{tot} - h_{H_2O})}{w} + \epsilon_W \frac{w h_{H_2O}}{w}$$

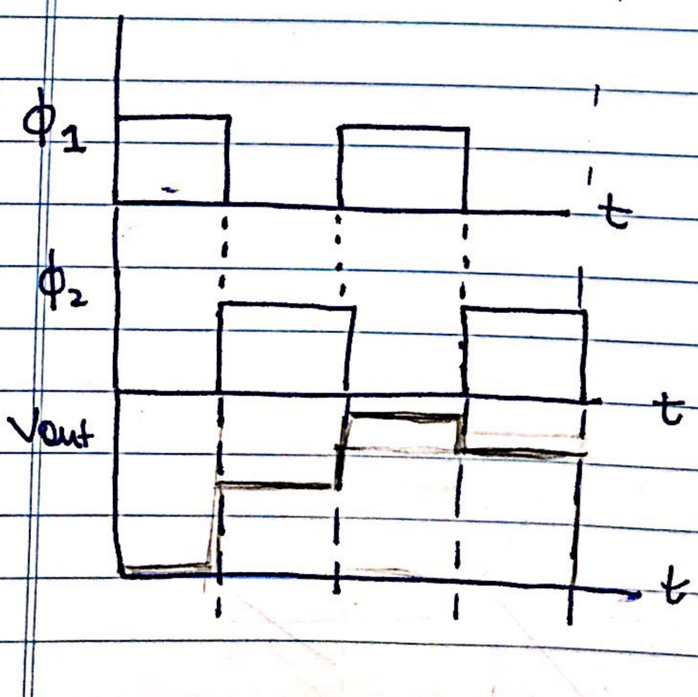
$$= \frac{\epsilon_0 \, w(h_{tot} - h_{H_2O})}{w} + \frac{\epsilon_W \, w h_{H_2O}}{w}$$

$$= \epsilon_0 (h_{tot} - h_{H_2O}) + 81 \epsilon_0 h_{H_2O}$$

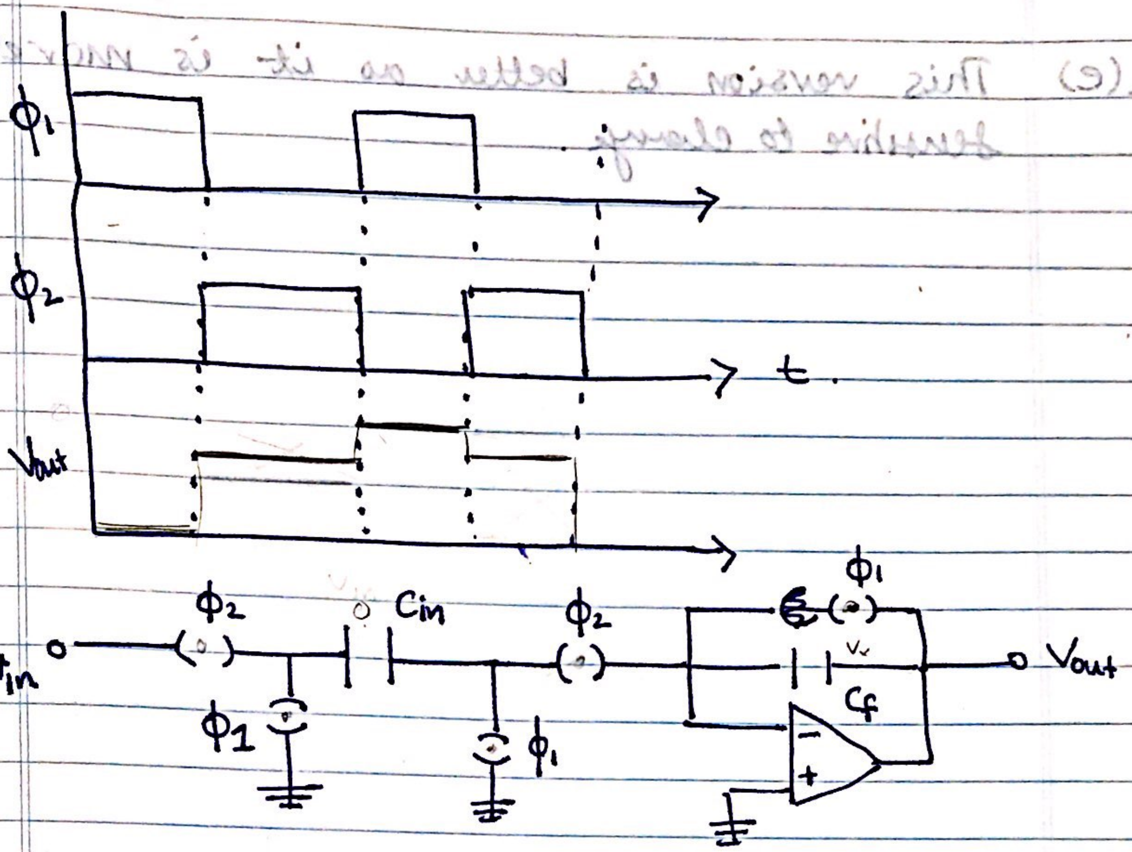
$$= C_{empty} - 80 \epsilon_0 h_{H_2O}$$

Q2(b)

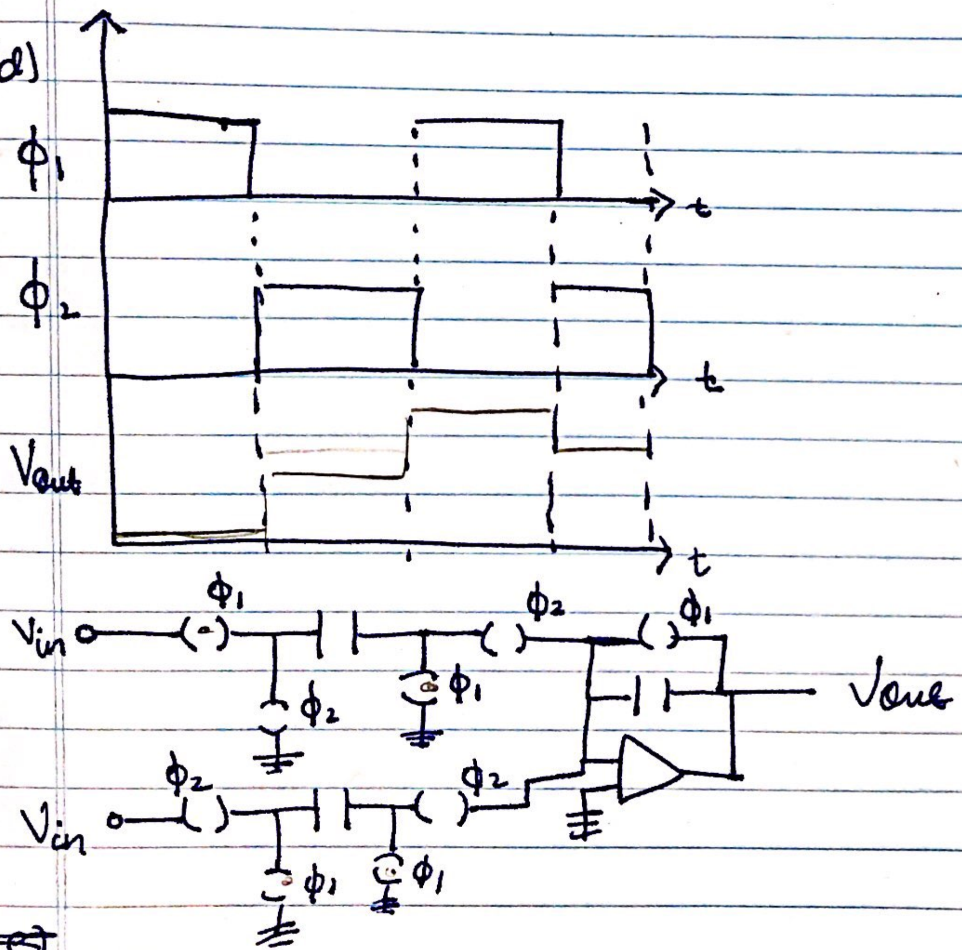




Q2 (c)


φ1
φ2
t
Vout
φ1
φ2
Cin
φ2
Vin
φ1
φ1
Cf
Vout

Q2 (d)


φ1
t
φ2
t
Vout
t
φ1
φ2
φ1
Vin
φ2
φ1
Vout
φ2
φ2
Vin
φ1
φ1

~~Q2(e)~~

8 Q2(e) This version is better as it is more sensitive to change.

Q3:Correlation

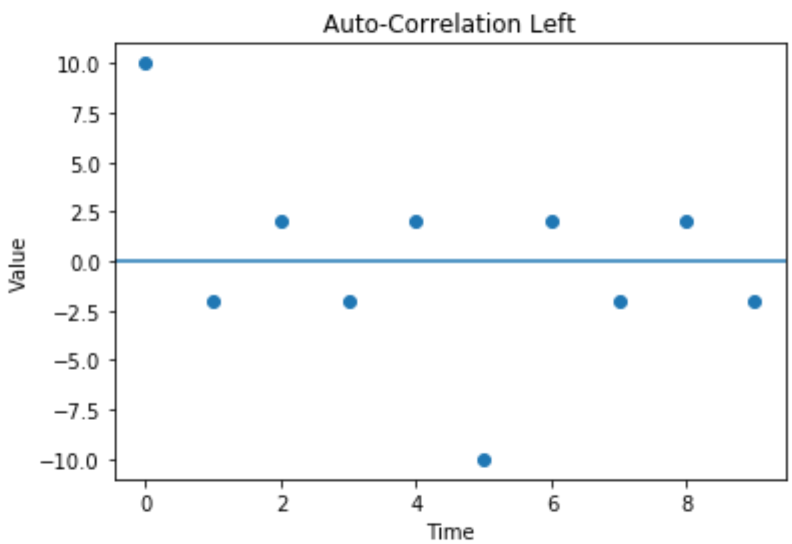
In [13]:
```
#Dependencies

import numpy as np
from scipy.linalg import circulant

#Graphing Dependencies
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.mlab as mlab
%matplotlib inline
```

Part (a) : Auto-correlation

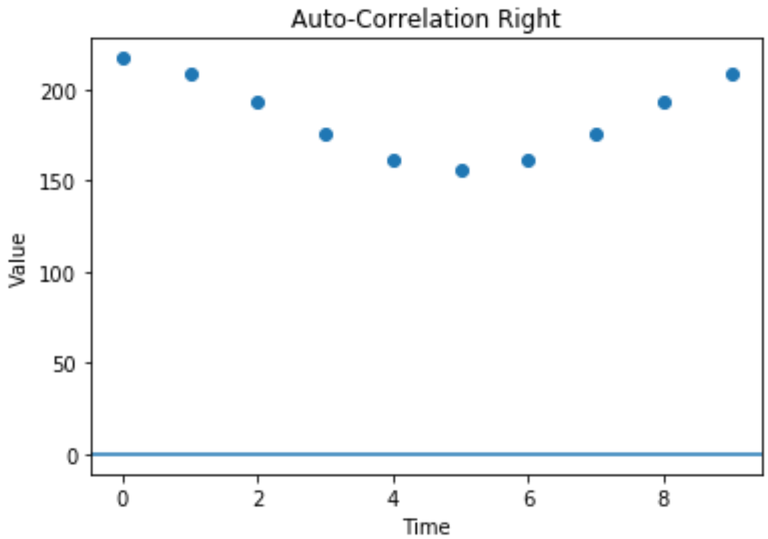
In [21]:

```
signal_left = np.array([1,-1,1,-1,-1,-1,1,-1,1,1]) #From the Image in the Qu
signal_left_circulant = circulant(signal_left)
auto_correlation_left = np.dot(signal_left, signal_left_circulant)

signal_right = np.array([1,2,3,4,5,6,7,6,5,4])
signal_right_circulant = circulant(signal_right)
auto_correlation_right = np.dot(signal_right, signal_right_circulant)

print("Auto-Correlation Left: ", auto_correlation_left)
# plt.plot(auto_correlation_left)
plt.scatter([i for i in range(0,len(auto_correlation_left))], auto_correlati
plt.axhline()
plt.xlabel("Time")
plt.ylabel("Value")
plt.title("Auto-Correlation Left")
plt.show()

print("Auto-Correlation Right: " , auto_correlation_right)
# plt.plot(auto_correlation_right)
plt.scatter([i for i in range(0,len(auto_correlation_right))], auto_correlat
plt.axhline()
plt.xlabel("Time")
plt.ylabel("Value")
plt.title("Auto-Correlation Right")
plt.show()
```

```
Auto-Correlation Left:  [ 10  -2   2  -2   2 -10   2  -2   2  -2]
```



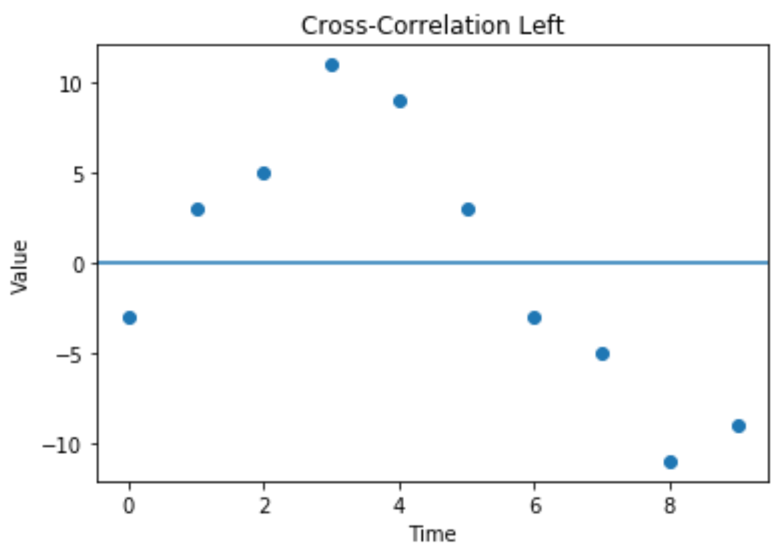

```
Auto-Correlation Right:  [217 208 193 176 161 156 161 176 193 208]
```



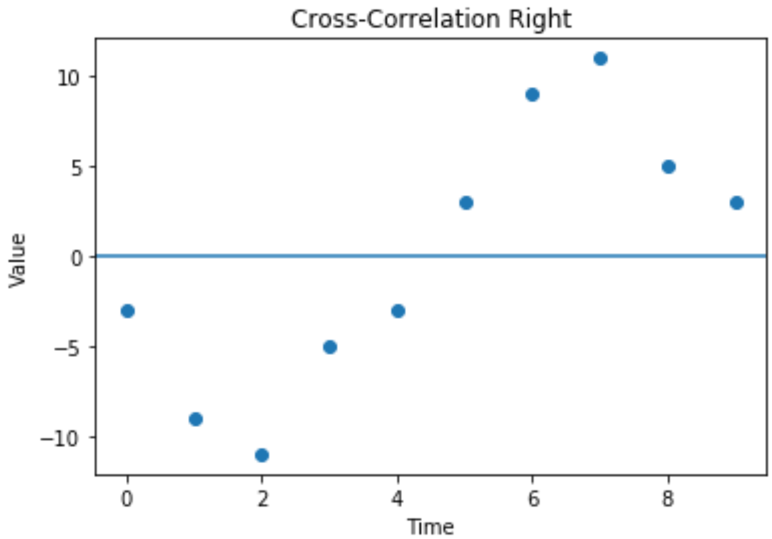
Part (b) : Cross-correlation

In [22]:
```
signal_left = np.array([1,-1,1,-1,-1,-1,1,-1,1,1]) #From the Image in the Qu
signal_left_circulant = circulant(signal_left)

signal_right = np.array([1,2,3,4,5,6,7,6,5,4])
signal_right_circulant = circulant(signal_right)

cross_correlation_left = np.dot(signal_left, signal_right_circulant)
cross_correlation_right = np.dot(signal_right, signal_left_circulant)

print("Cross-Correlation Left: ", cross_correlation_left)
plt.scatter([i for i in range(0,len(cross_correlation_left))], cross_correla
plt.axhline()
plt.xlabel("Time")
plt.ylabel("Value")
plt.title("Cross-Correlation Left")
plt.show()

print("Cross-Correlation Right: " , cross_correlation_right)
plt.scatter([i for i in range(0,len(cross_correlation_right))], cross_correl
plt.axhline()
plt.xlabel("Time")
plt.ylabel("Value")
plt.title("Cross-Correlation Right")
plt.show()
```

```
Cross-Correlation Left:  [ -3   3   5  11   9   3  -3  -5 -11  -9]
```




```
Cross-Correlation Right:  [ -3  -9 -11  -5  -3   3   9  11   5   3]
```



In [ ]:

Q4(a) The largest inner products seem to be ~~about~~ around 0.07 − 0.08 in value

Q4(b) We see the maximum correlation occurs at a shift of 10 (# np.argmax(corr))
$\therefore$ the delay's value is 10.

Reasoning
$\vec{s}_1^{(j)} \cdot \vec{s}_1^{(k)} = 1$ iff $k = j$ (which is satisfied at 10)

Q4(c) • $\|\langle \vec{s}_1, n \rangle\|$ is less than 0.10, around 0.03. (give or take)
• How does it compare to $\langle \vec{s}_1, \vec{s}^{(j)} \rangle$ from (a) ?
$\downarrow$
It is roughly the same

Q4(d) Yes we can.
Reasoning: $\langle \vec{s}_1^{(k)}, \vec{y} \rangle$ (from above) → $k = j$: the dot product is large
→ $k \neq j$: the dot product is small

$\therefore$ It works even for $k \neq j$

Q4(e) for $\vec{y} = \vec{s}_1^{(j)} + \vec{n}$

The graph generated is similar, and as we see argmax() returns 10 (which is roughly correct)

→

Q4 (e) Contd

for $\vec{y} = \vec{s}_1 + 10\vec{n}$

Each time we run the simulation, the shape of our graph changes and argmax() returns a different value, thus there is no clear maxima value of correlation.

Q4 (f) Yes this works!

Reason: $\vec{s}_2$ can be treated as manageable noise (the medium noise case : $\vec{y} = \vec{s}_1 + \vec{n}$ from (e))

Q4 (g) Since signal 1 > signal 2 (much larger)

$\therefore$ finding signal 1 works $\Rightarrow$ signal 2 is treated as low noise

but, finding signal 2 does not work $\Rightarrow$ signal 1 is treated as very high noise

Q4 (h) Yes this works!

Reason
- $\vec{y} = \vec{s}_1 + 0.1\vec{s}_2 - \vec{s}1$ (according to Question)

$\Rightarrow \vec{y} = 0.1\vec{s}_2$

$\therefore$ we can find $\vec{s}_2$

- We find $\vec{s}1$ ~~and~~ the same was as before.

Q4 (i) Yes this works!

Reason Assuming we have shift j correctly done:

$$\vec{s}_1^{(j)} \cdot \vec{y} = \vec{s}_1^{(j)} \cdot (\alpha_1 \vec{s}_1^{(j)} + \alpha_2 \vec{s}_2^{(k)}) \quad \# \text{ from Q}$$

$$= \alpha_1 \vec{s}_1^{(j)} \cdot \vec{s}_1^{(j)} + \alpha_2 \vec{s}_1^{(j)} \cdot \vec{s}_2^{(k)}$$

$$= \alpha_1 + \alpha_2 \vec{s}_1^{(j)} \cdot \vec{s}_2^{(k)} \quad [\vec{a} \cdot \vec{a} = 1]$$

$$\approx \alpha_1 \quad [\vec{s}_1^{(j)} \cdot \vec{s}_2^{(k)} \text{ very small}]$$

Scanned by CamScanner

Problem Set 11 Code

In [1]:
```
%pylab inline
import numpy as np
import matplotlib.pyplot as plt
```

```
Populating the interactive namespace from numpy and matplotlib
```

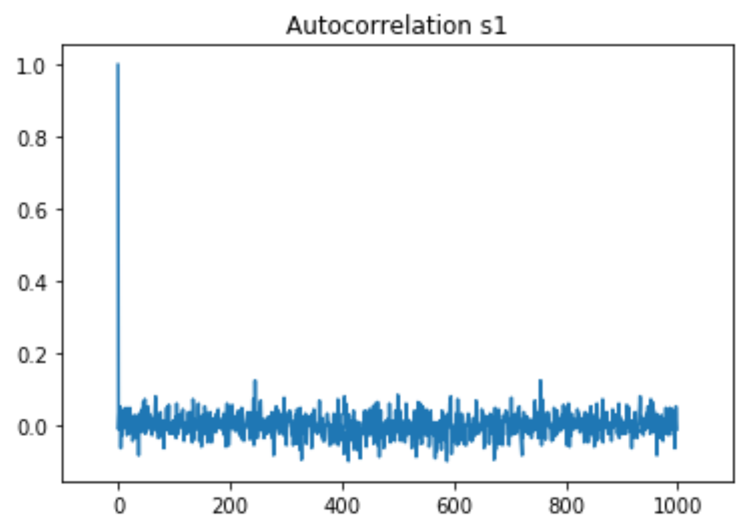
Finding Signals in Noise

In [2]:
```
# Run this first
%matplotlib inline
import numpy as np
import scipy as sp
import scipy.linalg as la
import pylab as plt
import numpy.random

N = 1000

def rand_vector(n): # returns a random {+1, -1} vector of length n
    return np.random.randint(2, size=n)*2 - 1.0

def rand_normed_vector(n): # returns a random normalized vector of length n
    x = rand_vector(n)
    return x / la.norm(x)

def cross_corr(f, g):
    # returns the cross-correlation (a vector of all the inner products of
    C = la.circulant(f)
    corr = C.T.dot(g)
    return corr
```

(a)

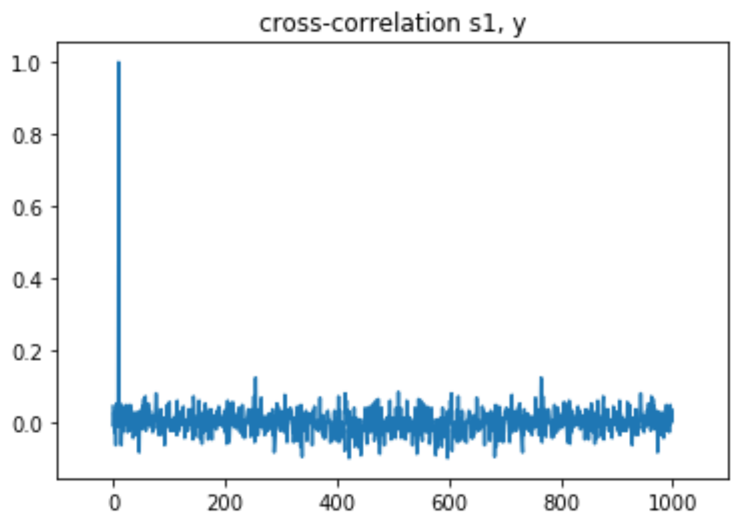
In [3]:
```
# generate a random normalized vector for s1
# (running this cell again will generate a new random vector)
s1 = rand_normed_vector(N)

# compute all the inner products of s1 with shifted versions of s1
# (ie, the cross-correlation of s1 with s1)
corr = cross_corr(s1, s1)

# The inner prouct <s1, s1^(1)> is:
print(corr[1])

# np.roll circularly shifts the signal
# so the above inner product could be computed as:
print(np.dot(s1, np.roll(s1,1)))

# Plot the autocorrelation:
plt.title("Autocorrelation s1")
plt.plot(corr)

x1,x2,y1,y2 = plt.axis()
plt.axis([x1-50,x2+50,y1,y2])

plt.show()
```

```
-0.012
-0.012
```



(b)

In [4]:
```
y = np.roll(s1, 10) # Received y = s1 shifted by 10

# Compute the cross-correlation (all the inner products of y with shifted ve
corr = cross_corr(s1, y)

# Plot
plt.title("cross-correlation s1, y")
plt.plot(corr)

x1,x2,y1,y2 = plt.axis()
plt.axis([x1-50,x2+50,y1,y2])
plt.show()

# Find the index of maximum correlation (inner product)
print(np.argmax(corr))
```




```
10
```

(c)

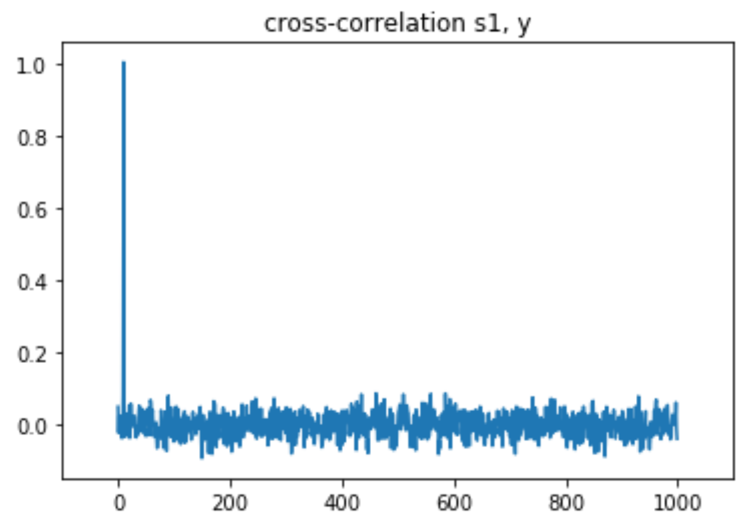
In [34]:
```
# generate a random normalized vector for s1,
# and a random normalized vector for n
# (running this cell again will generate new random vectors)
s1 = rand_normed_vector(N)
n = rand_normed_vector(N)

print(np.abs(np.dot(s1, n)))
```

```
0.038
```

(d)

This is the code from part (b), but with the received signal $\vec{y}$, which is corrupted by noise.

In [35]:
```
s1 = rand_normed_vector(N)
n = rand_normed_vector(N)
y = np.roll(s1, 10) + 0.1*n

corr = cross_corr(s1, y)

plt.title("cross-correlation s1, y")
plt.plot(corr)

x1,x2,y1,y2 = plt.axis()
plt.axis([x1-50,x2+50,y1,y2])
plt.show()

# Find the index of maximum correlation (inner product)
np.argmax(corr)
```

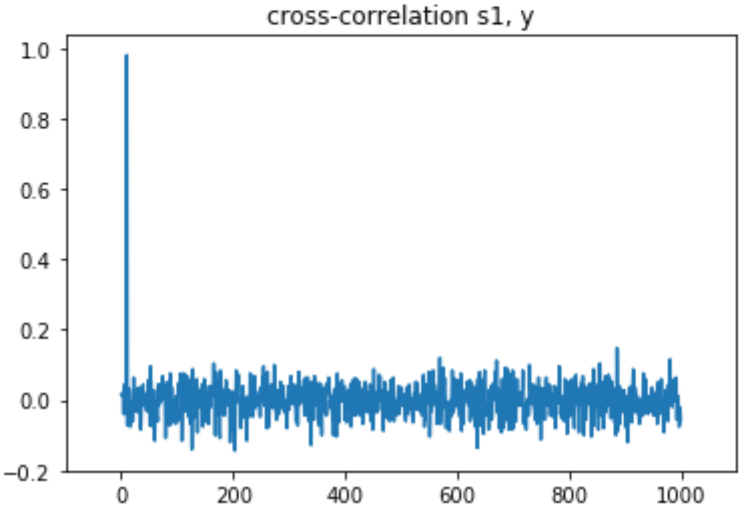


Out[35]: 10

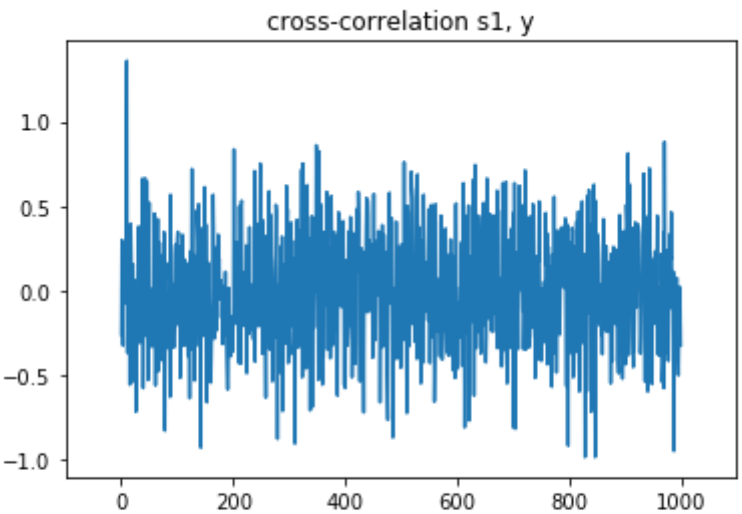
(e)

Copy the code provided for part (d), but modify it appropriately, so that the noise is higher. You should generate two cross-correlation plots, one for each noise level in the question. (You can just copy the code from part (d) twice.)

In [44]:
```
## CODE HERE
#HIGH
s1 = rand_normed_vector(N)
n = rand_normed_vector(N)
y = np.roll(s1, 10) + n

corr = cross_corr(s1, y)

plt.title("cross-correlation s1, y")
plt.plot(corr)

x1,x2,y1,y2 = plt.axis()
plt.axis([x1-50,x2+50,y1,y2])
plt.show()

# Find the index of maximum correlation (inner product)
print(np.argmax(corr))

#VERY HIGH
s1 = rand_normed_vector(N)
n = rand_normed_vector(N)
y = np.roll(s1, 10) + 10*n

corr = cross_corr(s1, y)

plt.title("cross-correlation s1, y")
plt.plot(corr)

x1,x2,y1,y2 = plt.axis()
plt.axis([x1-50,x2+50,y1,y2])
plt.show()

# Find the index of maximum correlation (inner product)
print(np.argmax(corr))
# max_val = corr[np.argmax(corr)]
# print("Max Val: ", max_val)
# count = 0
# for x in corr:
#     if(x == max_val):
#         count = count + 1
# print(count)
```



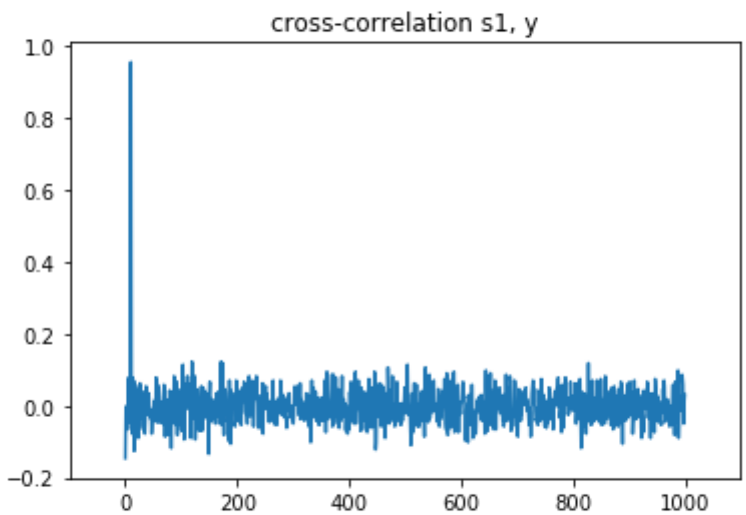

```
10
```

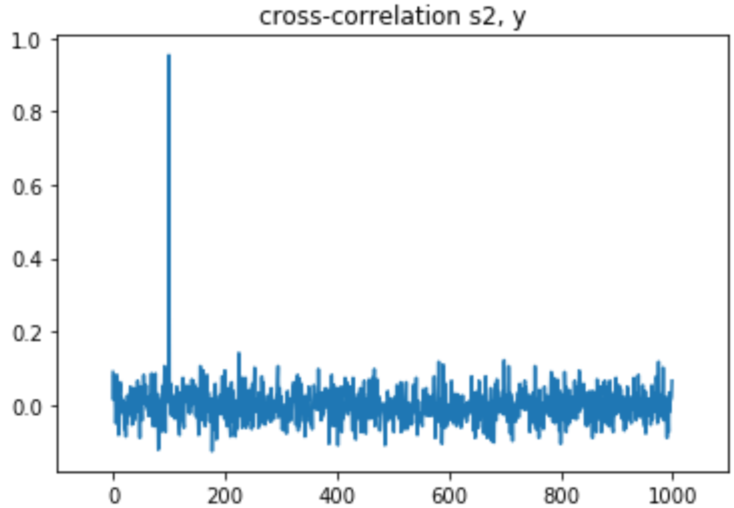



```
10
Max Val:  1.36
1
```

(f)

In [45]:
```
s1 = rand_normed_vector(N)
s2 = rand_normed_vector(N)

y = np.roll(s1, 10) + np.roll(s2, 100)

# Compute cross-correlations:
corr_s1_y = cross_corr(s1, y)
corr_s2_y = cross_corr(s2, y)

# Plot cross-correlations:
plt.title("cross-correlation s1, y")
plt.plot(cross_corr(s1, y))
x1,x2,y1,y2 = plt.axis()
plt.axis([x1-50,x2+50,y1,y2])
plt.show()

plt.title("cross-correlation s2, y")
plt.plot(cross_corr(s2, y))
x1,x2,y1,y2 = plt.axis()
plt.axis([x1-50,x2+50,y1,y2])
plt.show()

j = np.argmax(corr_s1_y) # find the first signal delay (max index of correl
k = np.argmax(corr_s2_y) # find the second signal delay
print(j,k)
```



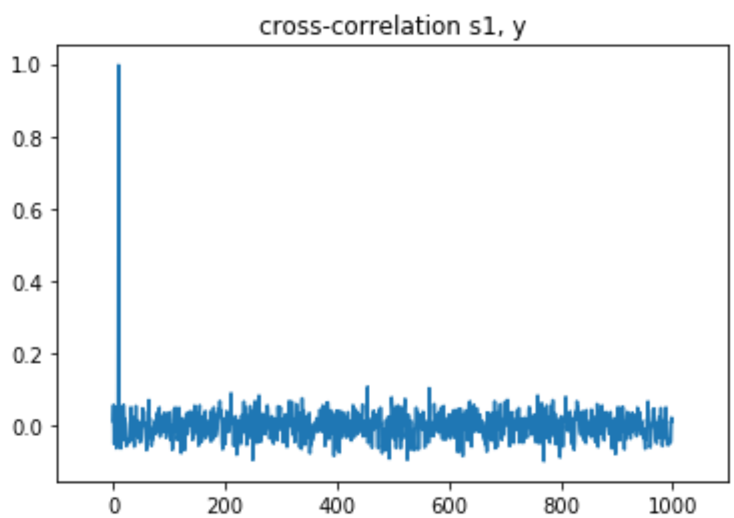


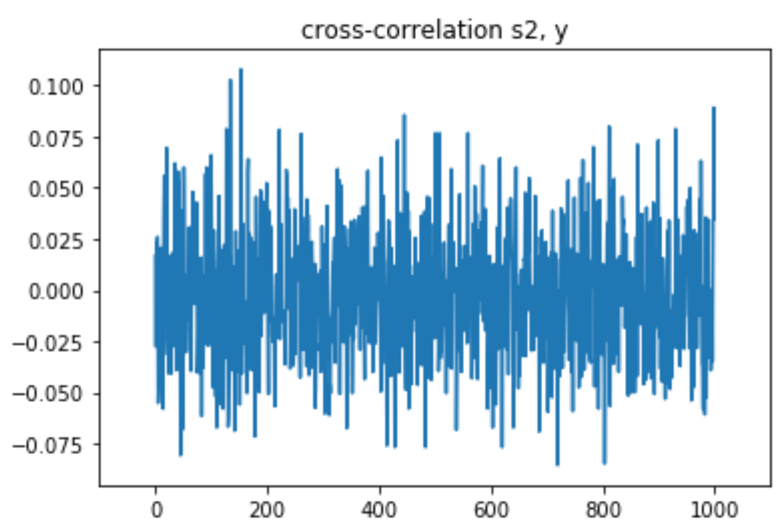

```
10 100
```

(g)

This is the same code as part (f), but with slight modification to how the received signal y generated. Run the below cell a few times to test for different choices of random signals.

In [48]:
```
s1 = rand_normed_vector(N)
s2 = rand_normed_vector(N)

y = np.roll(s1, 10) + 0.1*np.roll(s2, 100)

# Compute cross-correlations:
corr_s1_y = cross_corr(s1, y)
corr_s2_y = cross_corr(s2, y)

# Plot cross-correlations:
plt.title("cross-correlation s1, y")
plt.plot(cross_corr(s1, y))
x1,x2,y1,y2 = plt.axis()
plt.axis([x1-50,x2+50,y1,y2])
plt.show()

plt.title("cross-correlation s2, y")
plt.plot(cross_corr(s2, y))
x1,x2,y1,y2 = plt.axis()
plt.axis([x1-50,x2+50,y1,y2])
plt.show()
```



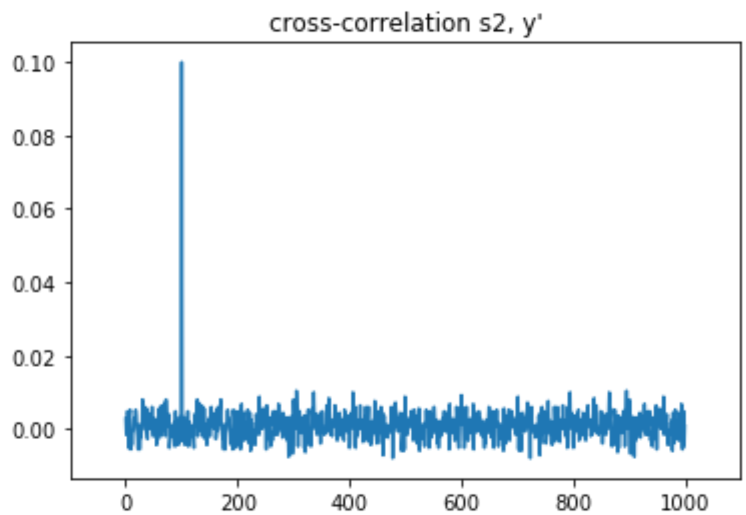


(h)

In [49]:

```
corr_s1_y = cross_corr(s1, y)
j = np.argmax(corr_s1_y) # find the first signal delay
print(j)

# subtract out the contribution of the first signal
y_prime = y - np.roll(s1, j)

# correlate the residual against the second signal
corr_s2_y = cross_corr(s2, y_prime)

# Plot
plt.title("cross-correlation s2, y'")
plt.plot(corr_s2_y)
x1,x2,y1,y2 = plt.axis()
plt.axis([x1-50,x2+50,y1,y2])
plt.show()

k = np.argmax(corr_s2_y) # find the second signal delay by looking at the in
print(k)
```

```
10
```




```
100
```

(i)

In [51]:
```
s1 = rand_normed_vector(N)
s2 = rand_normed_vector(N)

y = 0.7*np.roll(s1, 10) + 0.5*np.roll(s2, 100)

corr_s1_y = cross_corr(s1, y)
j = np.argmax(corr_s1_y) # find the first signal delay

corr_s2_y = cross_corr(s2, y)
k = np.argmax(corr_s2_y) # find the second signal delay

print(j, k)

# Once we have found the shifts, estimate the coefficients as inner products
a1 = np.dot(y, np.roll(s1, j))
a2 = np.dot(y, np.roll(s2, k))

print(a1, a2)
```

```
10 100
0.699 0.4986
```

(j)

This is the same code as part (i), but with noise added to the received signal $\vec{y}$.

In [54]:
```
s1 = rand_normed_vector(N)
s2 = rand_normed_vector(N)
n = rand_normed_vector(N)

y = 0.7*np.roll(s1, 10) + 0.5*np.roll(s2, 100) + 0.1*n

corr_s1_y = cross_corr(s1, y)
j = np.argmax(corr_s1_y) # find the first signal delay

corr_s2_y = cross_corr(s2, y)
k = np.argmax(corr_s2_y) # find the second signal delay

print(j, k)

# Once we have found the shifts, estimate the coefficients as inner products
a1 = np.dot(y, np.roll(s1, j))
a2 = np.dot(y, np.roll(s2, k))

print(a1, a2)
```

```
10 100
0.6662 0.4518
```

In [ ]:

In [ ]:

Q4(j) The estimates are in the range of ± 0.04

Q5 We all worked individually and came together to discuss when we got stuck.

Siddharth Mehta	25118777
Mohan Lakshman	24199173
Jack Sullivan	25195800
James Jiang	25236910