تمرین سوم مدارهای منطقی برنامه‌پذیر

هم‌بستگی متقابل (Cross-correlation)

الگوریتم هم‌بستگی متقابل (*Cross-correlation*)، به طور گسترده در حوزه‌های مختلف پردازش سیگنال‌های دیجیتال به کار گرفته می‌شود. در این نوع از هم‌بستگی، سیگنال ورودی با سیگنالی دیگر هم‌بسته می‌شوند تا میزان شباهت بین آن‌ها محاسبه گردد. تعریف کلی همبستگی متقابل دو سیگنال y و x به صورت زیر است:

$$R_{xy}(\tau) = \int_{-\infty}^{+\infty} x(t)y^*(t-\tau)dt$$

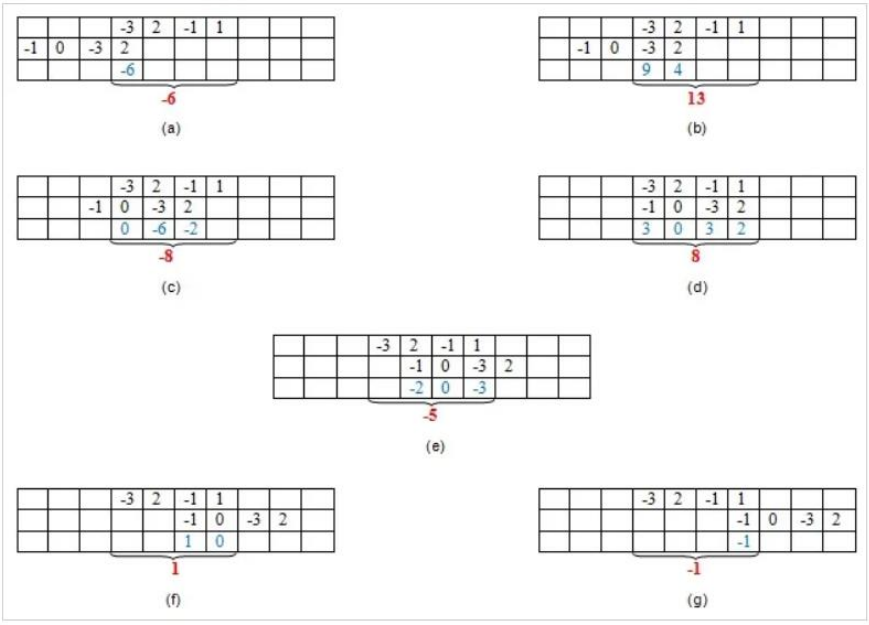
در صورتی که سیگنال‌های y و x را به صورت گسسته و حقیقی در نظر بگیریم، الگوریتم *Cross-correlation* را می‌توان به صورت زیر تعریف کرد:

$$R_{xy}[m] = \sum_{n=-\infty}^{+\infty} x[n]y[n-m]$$

Or

$$R_{xy}[m] = \sum_{n=-\infty}^{+\infty} x[n+m]y[n]$$

نحوه‌ی محاسبه‌ی *cross-correlation* برای دو سیگنال دیجیتال و حقیقی *x={-1, 0, -3, 2}* و *y={-3, 2, -1, 1}* در شکل زیر نمایش داده شده است. با توجه به اینکه طول این دو سیگنال محدود است، طول *cross-correlation* نیز محدود خواهد بود.



شکل (۱) محاسبه *cross-correlation* برای دو سیگنال *x , y*

ردیف اول جدول مربوط به سیگنال *x* و ردیف دوم مربوط به سیگنال *y* است. نمونه‌هایی که به صورت آبی در ردیف سوم نمایش داده شده از ضرب مقادیر دو ردیف اول محاسبه می‌شود. مقادیر زیر براکت هم از جمع مقادیر آبی محاسبه می‌شود که در حقیقت، نمونه‌های *cross-correlated* دو سیگنال را نمایش می‌دهند.

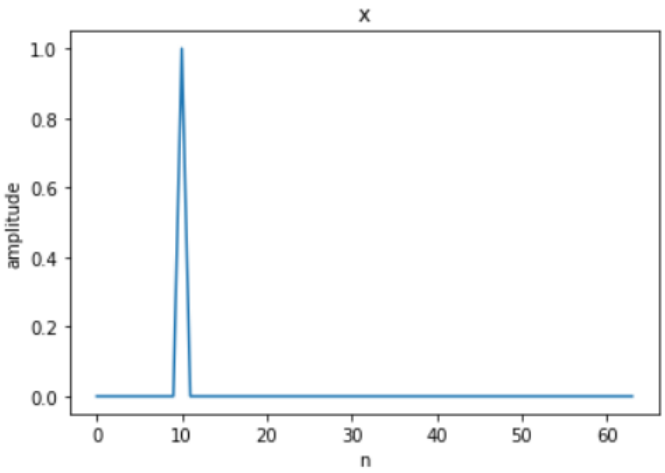
R_{xy} (مقادیر همبستگی متقابل این دو سیگنال) به صورت *{-6, 13, -8, 8, -5, 1, -1}* محاسبه شده که مقدار ۸ در آن مربوط به $R_{xy}[0]$ است. در این مثال، زمانی که دو نمونه‌ی آخر *y[n]* با دو نمونه‌ی ابتدایی *x[n]* هم‌پوشانی دارند، سیگنال همبسته‌شده بیشترین مقدار $(R_{xy}[-2] = 13)$ را دارد. زیرا در این حالت سیگنال دوم بهترین هم‌پوشانی را با سیگنال اول دارد و دو نمونه ی هر دو سیگنال در این حالت یکسان هستند. با توجه به این موضوع به نظر می‌رسد زمانی مقدار *cross-correlation* به حداکثر خود می‌رسد که دو سیگنال بیشترین شباهت را به یکدیگر داشته باشند. جهت مشاهده‌ی توضیحات بیشتر و کاربردهای این الگوریتم می‌توانید به لینک زیر مراجعه نمایید:

https://www.allaboutcircuits.com/technical-articles/understanding-correlation

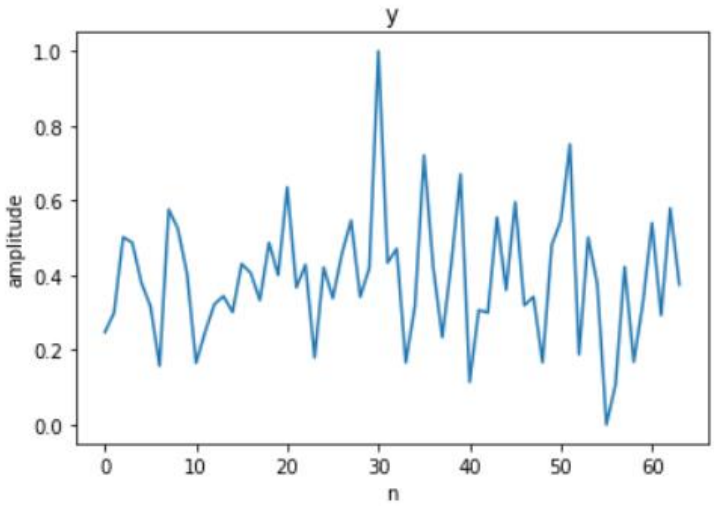
شرح تمرین

هدف از این تمرین، پیاده‌سازی الگوریتم *cross-correlation* است. در این پیاده‌سازی، ورودی دو سیگنال گسسته‌ی حقیقی و با طول محدود *x* و *y* است و خروجی، مقدار تاخیر زمانی *(Time Delay or Lag)* و سیگنال همبسته‌ی متقابل R_{xy} می‌باشد.

فرض کنید سیگنال *x*، سیگنال پالس واحدی است که از طریق یک فرستنده برای اندازه‌گیری فاصله از مانع روبه‌رویش فرستاده شده است. این سیگنال پس از برخورد به مانع بازتاب شده و سیگنال حاوی نویز *y* توسط گیرنده دریافت شده است. با توجه به اینکه سیگنال بازتاب شده از جنس سیگنال فرستاده شده است، شما می‌توانید با استفاده اختلاف زمانی پیک‌های این دو سیگنال و چند پارامتر دیگر فاصله‌ی فرستنده–گیرنده را با مانع رو به روی آن محاسبه کنید. سیگنال‌های *x* و *y* طول محدود ۶۴ نمونه ای دارند و در شکل زیر قابل مشاهده هستند.



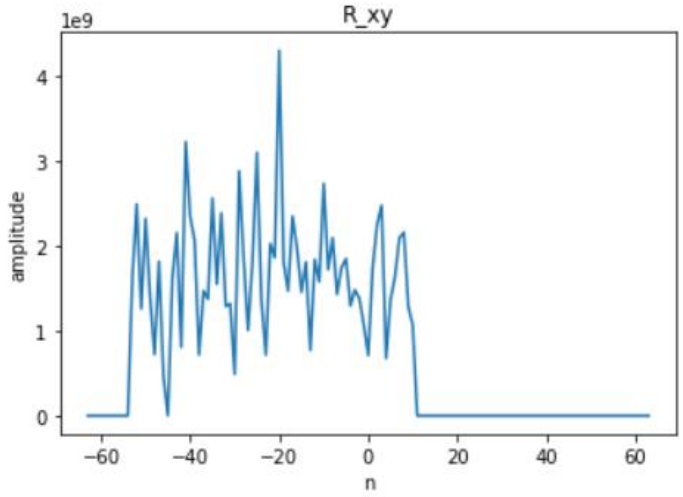
شکل (۲) سیگنال *x*



شکل (۳) سیگنال y

با مقایسه‌ی پیک دو سیگنال x و y مشاهده می‌شود که این دو سیگنال با یکدیگر ۲۰ واحد زمانی اختلاف دارند.

این دو سیگنال طول محدودی دارند. در نتیجه، طول سیگنال همبسته‌ی خروجی نیز محدود خواهد بود و فرمول ذکر شده در ابتدای صورت تمرین برای m های کوچکتر از ۶۳- و بزرگتر از ۶۳ غیر معتبر خواهد بود. با توجه به این موضوع، سیگنال *cross-correlated* برای دو سیگنال x و y محاسبه شده که در شکل (۴) قابل مشاهده است.



شکل (۴) سیگنال R_{xy}

همانطور که در شکل بالا قابل مشاهده است، سیگنال R_{xy} در نقطه‌ی ۲۰- به حداکثر مقدار خود می‌رسد که برابر اختلاف زمانی بین سیگنال‌های x و y است.

قدم‌های زیر برای انجام این تمرین باید طی شود:

۱. پیاده‌سازی الگوریتم *cross-correlator* به صورت ترتیبی
   a. با توجه به اینکه در صورت تمرین هیچ طرحی *(Design)* پیشنهاد نشده است، از طرح پیشنهادی خود استفاده کنید.
   b. ورودی‌های *x* و *y* را ۱۶ بیتی، خروجی R_{xy} را ۳۲ بیتی و عرض بیت خروجی *Lag* (اختلاف زمانی) را بر مبنای طول سیگنال در نظر بگیرید.
   c. از حافظه‌ی *RAM* برای *Buffer* کردن سیگنال‌های ورودی *x* و *y* و خروجی R_{xy} استفاده کنید. (در الگوریتم‌هایی که برای انجام پردازش به تمام نمونه‌های سیگنال احتیاج دارند، بافر کردن با استفاده از *RAM* امری معمول است.)
   d. پس از اتمام محاسبات، نمونه‌های R_{xy} از *Buffer* خوانده شده و در خروجی قرار می‌گیرد.
۲. تولید سیگنال *x* و *y* با طول دلخواه (برای اینکار می‌توانید از کد پایتون پیوست استفاده کنید.)
۳. قرار دادن سیگنال‌های *x* و *y* درون حافظه‌ی *ROM* (از کدی که در کلاس برای تعریف حافظه‌ی *ROM* توضیح داده شد، استفاده کنید.)
۴. نوشتن تست بنچی که ۶۴ نمونه‌ی سیگنال‌های *x* و *y* را به ترتیب ازحافظه *ROM* خوانده و آن‌ها را به ورودی *cross-correlator* بدهد و خروجی‌های مورد نظر (*Lag* و *R_xy*) را تولید کند.

نکات تکمیلی

- گزارش ارسالی باید شامل موارد زیر باشد:
  - توضیحات طرح پیشنهادی و قدم‌های انجام شده
  - سطح مصرفی *FPGA* (منابع مصرفی: تعداد *Slice*، *FF* و ...) و حداکثر فرکانس کاری
  - تصویری از نتایج شبیه‌سازی نهایی (ترجیحا از شبیه‌ساز *Vivado* استفاده کنید)
- کلیه کدهای نوشته شده در آخر گزارش به صورت تک ستونی آورده شود.
- سعی کنید که کدهای *VHDL* تا حد ممکن دارای *Comment* بوده و به صورت مرتب نوشته شود.
- کلیه‌ی کدهای *VHDL* به همراه گزارش با فرمت‌های *Word* و *PDF* به صورت یک فایل *zip*. آرشیو شده و در سامانه‌ی *Courses* آپلود گردد.
- در صورت امکان، سعی کنید که از نرم‌افزار *Vivado* برای انجام این تمرین استفاده کنید.
- نیازی به ارسال پروژه‌ی کامل نیست و فقط فایل‌های *VHDL* را آپلود نمایید.
- از کپی کردن به شدت بپرهیزید. در صورتی که دو گزارش دقیقا مشابه یکدیگر باشند، به هر دو گزارش نمره صفر داده خواهد شد.

موفق باشید.

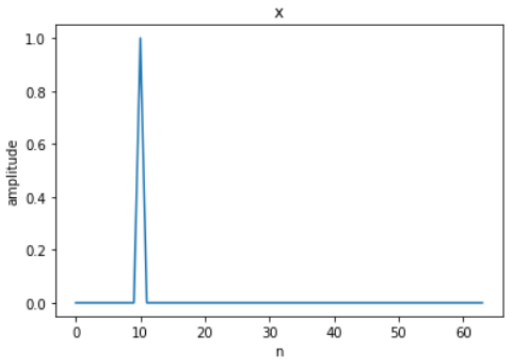
پیوست

```
#import libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

%matplotlib inline
```

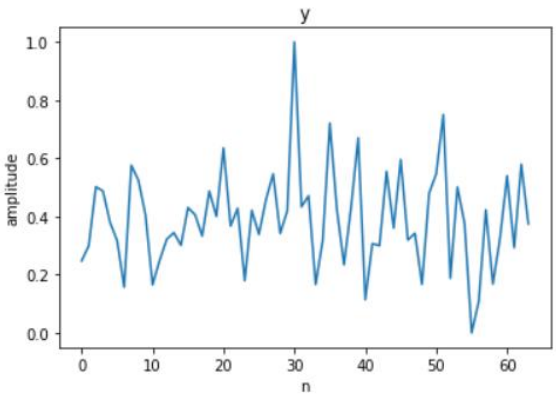
```
# signal length
length = 64

# time offest of each signal
t0 = 10
t1 = 30

# random white gaussian noise with variance = 0.1
noise = np.sqrt(0.06) * np.random.randn(length,1).squeeze()

# x is unit impulse
x = signal.unit_impulse(length, idx=t0)

# y is shifted noisy signal
y = signal.unit_impulse(length, idx=t1) + noise

# normalize y
y = (y - np.min(y)) / (np.max(y) - np.min(y))

# fixed-point signals
x = np.floor(x*(2**16-1))
y = np.floor(y*(2**16-1))

# print them out
print(f"x: {x}")
print(f"y: {y}")
```

```
x: [ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 65535. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
                    0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

y: [16211. 19592. 32884. 31927. 24911. 20760. 10297. 37742. 34366. 26564.
10823. 16319. 21075. 22549. 19737. 28189. 26581. 21804. 31909. 26225. 41658.
24057. 28053. 11763. 27552. 22179. 30030. 35801. 22415. 27461. 65535. 28390.
30856. 10894. 20789. 47225. 27703. 15337. 28379. 43911. 7469. 20090. 19638.
36361. 23601. 39041. 20924. 22412. 10910. 31440. 35780. 49150. 12295. 32827.
         24653. 0. 7043. 27666. 10988. 21301. 35368. 19189. 37936. 24546.]
```

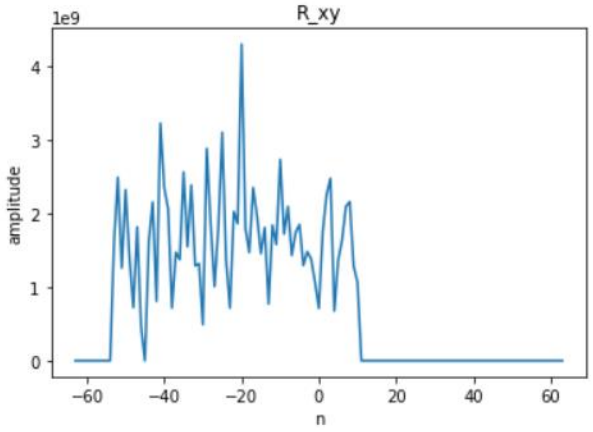
```
# plot first signal
plt.figure()
plt.plot(x/(2**16-1))
plt.title("x")
plt.ylabel("amplitude")
plt.xlabel("n")
plt.show()
```




```
# plot second signal
plt.figure()
plt.plot(y/(2**16-1))
plt.title("y")
plt.ylabel("amplitude")
plt.xlabel("n")
plt.show()
```




```
# calculating cross-corellation and time difference between signals
r = signal.correlate(x, y, mode='full', method='direct')
lag = np.argmax(r)

# plotting cross-corellated signal
t = np.linspace(-63, 63, 127)
plt.figure()
plt.plot(t, r)
plt.title("R_xy")
plt.ylabel("amplitude")
plt.xlabel("n")
plt.show()
print(f"time difference between two signals: {t[lag]}")
```




```
time difference between two signals: -20.0
```