



دانشگاه صنعتی خواجه نصیرالدین طوسی  
دانشکده مهندسی برق - گروه مهندسی کنترل

به نام خدا

دانشگاه تهران - دانشکده صنعتی خواجه نصیرالدین طوسی تهران

دانشکده مهندسی برق و کامپیوتر



## شبکه عصبی مک کلاچ - پیتز

نام و نام خانوادگی	محمد جواد احمدی
شماره دانشجویی	۴۰۱۰۰۰۸۶

## فهرست مطالب

۴	۱	ماشین متناهی قطعی (DFA)
۴	۱.۱	پاسخ قسمت الف
۴	۲.۱	پاسخ قسمت ب
۷	۳.۱	پاسخ قسمت ج
۱۰	۴.۱	پاسخ قسمت د
۱۴	۵.۱	راه حل و نگاه دوم

## فهرست تصاویر

۵	.....	نحوه ترکیب ورودی‌ها برای تولید D	۱
۶	.....	نحوه ترکیب ورودی‌ها برای تولید E	۲
۷	.....	نحوه ترکیب ورودی‌ها برای تولید F	۳
۸	.....	شبکه‌ای که می‌تواند برای خروجی D در نظر گرفته شود	۴
۸	.....	شبکه‌ای بهینه که می‌تواند برای خروجی D در نظر گرفته شود	۵
۹	.....	شبکه‌ای که می‌تواند برای خروجی E در نظر گرفته شود	۶
۹	.....	شبکه‌ای که می‌تواند برای خروجی F در نظر گرفته شود	۷
۹	.....	شبکه‌ای که می‌تواند برای خروجی F در نظر گرفته شود	۸
۱۰	.....	شبکه ادغام‌شده	۹
۱۰	.....	شبکه ادغام‌شده بهینه	۱۰
۱۱	.....	شبکه ادغام‌شده بهینه	۱۱
۱۵	.....	تصویر مربوط به حالت اول در نگاه دوم	۱۲
۱۶	.....	تصویر مربوط به حالت دوم در نگاه دوم	۱۳
۱۷	.....	تصویر مربوط به حالت سوم در نگاه دوم	۱۴

## فهرست جداول

۴	..... نمایش جدید جدول انتقال حالت	۱
---	-----------------------------------	---

## پرسش ۱. شبکه عصبی Mcculloch-Pitts

### ۱ ماشین متناهی قطعی (DFA)

#### توضیح پوشه کدهای Mcculloch-Pitts

کدهای مربوط به این قسمت، علاوه بر پوشه محلی کدها در این لینک آورده شده است.

#### ۱.۱ پاسخ قسمت الف

جدول انتقال حالتی که مطابق توضیحات سوال و متناسب با شبکه نوروها به صورت زیر است:

جدول ۱: نمایش جدید جدول انتقال حالت

IN			OUT		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	1	1
1	0	1	0	1	0
1	1	0	1	1	1
1	1	1	1	1	1

#### ۲.۱ پاسخ قسمت ب

در این قسمت براساس نام گذاری انجام شده در **جدول ۱**، به دست می آوریم که هر خروجی معادل چه ترکیبی از ورودی هاست. در جدول کارنو، روش SOP و روش POS دو روش مختلف برای تبدیل یک عبارت منطقی به یک عبارت استاندارد می باشند. در روش SOP (Sum of Products)، عبارت منطقی به صورت جمع بیشترین (OR) حاصل ضرب کوچکترین (AND) های ممکن تشکیل می شود. به عبارت دیگر، ابتدا برای هر حالت ممکن مقادیر ورودی، یک خروجی (AND) از متغیرهای ورودی ایجاد می شود. سپس تمام این خروجی ها با هم جمع می شوند. در روش POS (Product of Sums)، عبارت منطقی به صورت ضرب کوچکترین (AND) حاصل جمع بیشترین (OR) های ممکن تشکیل می شود. به عبارت دیگر، ابتدا برای هر حالت ممکن مقادیر ورودی، یک جمع (OR) از متغیرهای ورودی ایجاد می شود. سپس تمام این جمع ها با هم ضرب می شوند. استفاده از هر

دوروش، به دلیل وجود یک تعداد محدودی ورودی، به راحتی قابل انجام است. با ارائه این توضیحات، ابتدا به سراغ خروجی D می‌رویم. جداول و محاسبات مربوط به این خروجی به شرح زیر است (منطق SOP):

$\bar{C}$	$C$	$\bar{C}$	$C$				
$\bar{A} \cdot \bar{B}$	0	0	$\bar{A} \cdot \bar{B}$	0	1	(2, 6)	B. $\bar{C}$
$\bar{A} \cdot B$	1	0	$\bar{A} \cdot B$	2	3	(4, 6)	A. $\bar{C}$
A.B	1	1	A.B	6	7	(6, 7)	A.B
A. $\bar{B}$	1	0	A. $\bar{B}$	4	5		

(۱)

با منطق POS اما داریم:

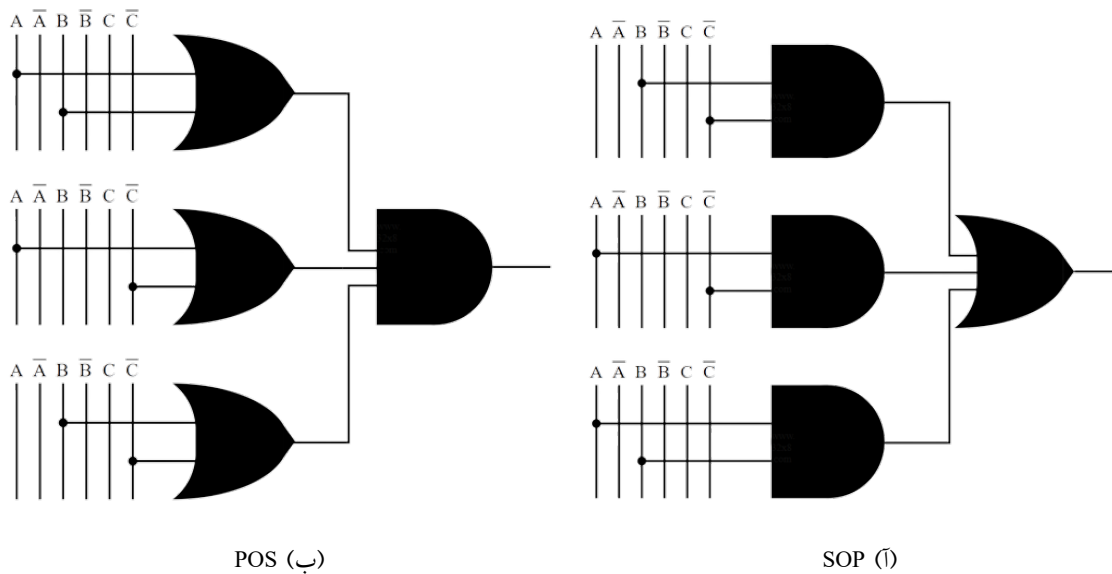
$\bar{C}$	$C$	$\bar{C}$	$C$		
$\bar{A} \cdot \bar{B}$	0	0	$\bar{A} \cdot \bar{B}$	0	1
$\bar{A} \cdot B$	1	0	$\bar{A} \cdot B$	2	3
$A \cdot B$	1	1	$A \cdot B$	6	7
$A \cdot \bar{B}$	1	0	$A \cdot B$	4	5

(0,1)	$\bar{A} \cdot \bar{B}$
(1,3)	$\bar{A} \cdot C$
(1,5)	$\bar{B} \cdot C$

(۲)

در نتیجه داریم (شکل ۱):

$$D = B\bar{C} + A\bar{C} + AB, \quad D = (A + B)(A + \bar{C})(B + \bar{C}) \quad (۳)$$



شکل ۱: نحوه ترکیب ورودی‌ها برای تولید D.

در ادامه به سراغ خروجی E می‌رویم. جداول و محاسبات مربوط به این خروجی به شرح زیر است (منطق SOP):

$\bar{C}$	C	$\bar{C}$	C		
$\bar{A} \cdot \bar{B}$	0	1	$\bar{A} \cdot \bar{B}$	0	1
$\bar{A} \cdot B$	0	1	$\bar{A} \cdot B$	2	3
$A \cdot B$	1	1	$A \cdot B$	6	7
$A \cdot \bar{B}$	1	1	$A \cdot \bar{B}$	4	5

(1, 3, 5, 7)	C
(4, 5, 6, 7)	A

(۴)

با منطق POS اما داریم:

$\bar{C}$	$\bar{C}$	$\bar{C}$	C		
$\bar{A} \cdot \bar{B}$	0	1	$\bar{A} \bar{B}$	0	1
$\bar{A} \cdot B$	0	1	$\bar{A} \cdot B$	2	3
$A \cdot B$	1	1	$A \cdot B$	6	7
$A \cdot \bar{B}$	1	1	$A \cdot \bar{B}$	4	5

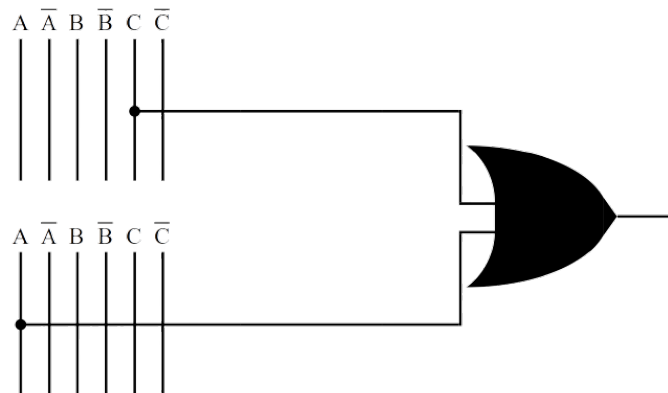
(0, 2)	$\bar{A} \cdot \bar{C}$
--------	-------------------------

(۵)

در نتیجه داریم (شکل ۲):

$$E = C + A \quad (۶)$$

در پایان هم به سراغ خروجی F می‌رویم. جداول و محاسبات مربوط به این خروجی به شرح زیر است (منطق SOP):



شکل ۲: نحوه ترکیب ورودی‌ها برای تولید E.

$\bar{C}$	C	$\bar{C}$	C		
$\bar{A} \cdot \bar{B}$	0	0	$\bar{A} \bar{B}$	0	1
$\bar{A} \cdot B$	0	0	$A \cdot B$	2	3
$A \cdot B$	1	1	$A \cdot B$	6	7
$A \cdot \bar{B}$	1	0	$A \cdot \bar{B}$	4	5

(4, 6)	$A \cdot \bar{C}$
(6, 7)	$A \cdot B$

(۷)

با منطق POS اما داریم:

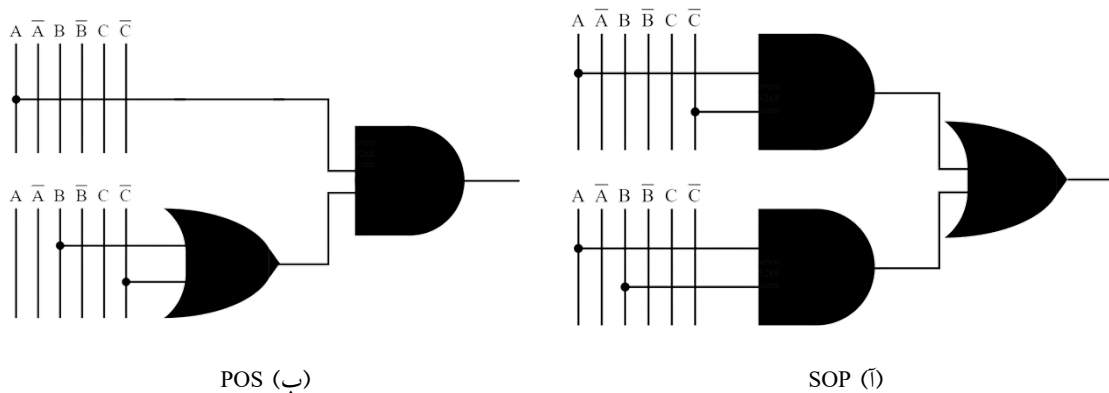
$\bar{C}$	$C$	$\bar{C}$	$C$		
$\bar{A} \cdot \bar{B}$	0	0	$\bar{A} \cdot \bar{B}$	0	1
$\bar{A} \cdot B$	0	0	$\bar{A} \cdot B$	2	3
$A \cdot B$	1	1	$A \cdot B$	6	7
$A \cdot \bar{B}$	1	0	$A \cdot \bar{B}$	4	5

(0, 1, 2, 3)	$\bar{A}$
(1, 5)	$\bar{B} \cdot C$

(۸)

در نتیجه داریم (شکل ۳):

$$F = A\bar{C} + AB \quad F = (A)(B + \bar{C}) = (A)(\overline{CB}) \quad (۹)$$



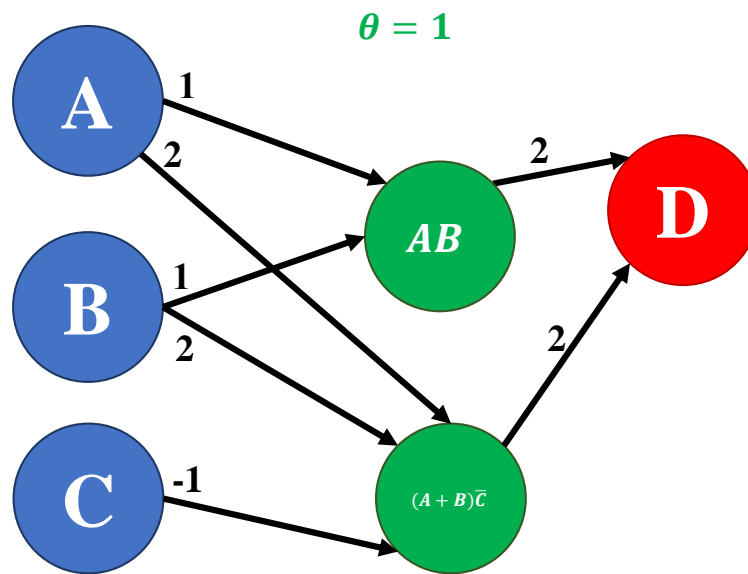
شکل ۳: نحوه ترکیب ورودی‌ها برای تولید F.

حال شبکه مربوط به هر خروجی را تشکیل می‌دهیم. ابتدا برای خروجی D و بر اساس رابطه ۳ شبکه آورده شده در شکل ۴ را تشکیل می‌دهیم. می‌توانستیم حالت دیگر را نیز در نظر بگیریم اما در آن صورت به تعداد نوروں بیش‌تری نیاز بود. هم‌چنین می‌توان حالت بهینه نشان‌داده شده در شکل ۵ را برای خروجی D در نظر گرفت. هرچند بهینگی مدنظر این قسمت از سوال نیست. در ادامه برای خروجی E و بر اساس رابطه ۶ شبکه آورده شده در شکل ۶ را تشکیل می‌دهیم. در پایان، برای خروجی F و بر اساس رابطه ۹ شبکه آورده شده در شکل ۷ را تشکیل می‌دهیم. می‌توانستیم حالت دیگر را نیز در نظر بگیریم اما در آن صورت به تعداد نوروں بیش‌تری نیاز بود. برای جلوگیری از تولید عدد دو برای F ناچاریم که شبکه را به شکل ۸ تغییر دهیم.

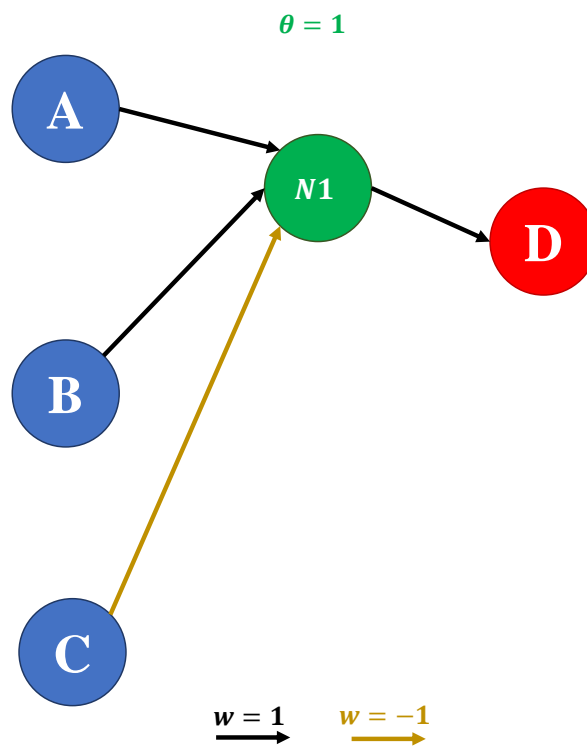
### ۳.۱ پاسخ قسمت ج

با ادغام شبکه‌های بهینه در نظر گرفته شده در شکل ۴، شکل ۶ و شکل ۷، شبکه نمایش داده شده در شکل ۹ را به دست می‌آوریم. با این وجود حالت بهینه و پاسخگوی این ادغام با سعی و خطا در پاسخ قسمت د به دست آمده است که آن را در شکل ۱۰ نمایش داده‌ایم (با فرض اینکه خروجی دارای تابع فعال‌ساز است و یا جنس صفر و یک آن از پیش تعریف شده و به محض دریافت یک، یک می‌شود). با این حال اگر اتصال مستقیم ورودی و خروجی را مجاز ندانیم، یک نوروں میانی برای خروجی E هم در نظر می‌گیریم و نتیجه به صورتی خواهد بود که در شکل ۱۱ نمایش داده شده است. هرچند در کد می‌توان این محدودیت را دور زد؛ اما به لحاظ منطق تئوری نیاز به استفاده از ۴ نوروں داریم تا خروجی بیش‌تر از یک نشود (آستانه در تمام حالات یک است).

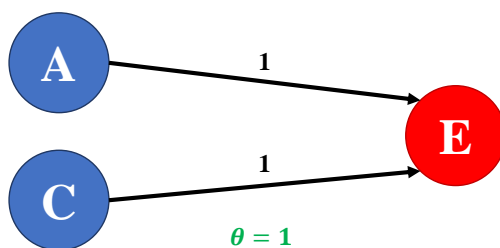




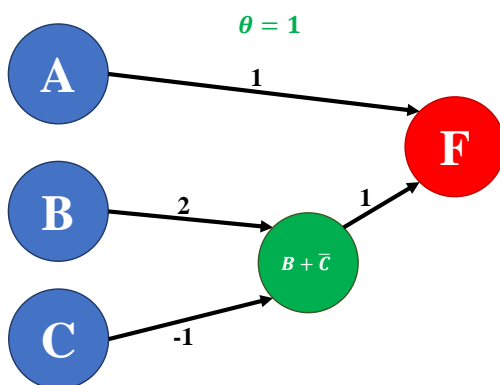
شکل ۴: شبکه‌ای که می‌تواند برای خروجی D در نظر گرفته شود.



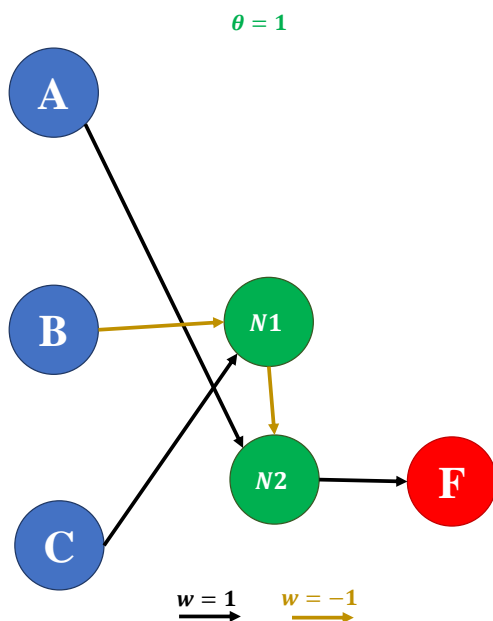
شکل ۵: شبکه‌ای بهینه که می‌تواند برای خروجی D در نظر گرفته شود.



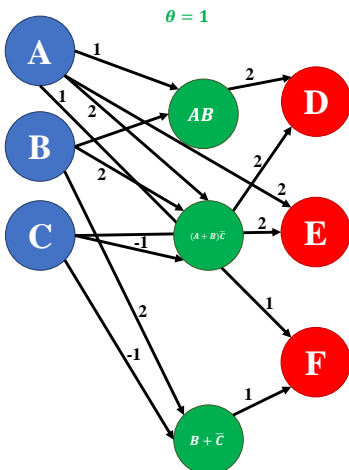
شکل ۶: شبکه‌ای که می‌تواند برای خروجی E در نظر گرفته شود.



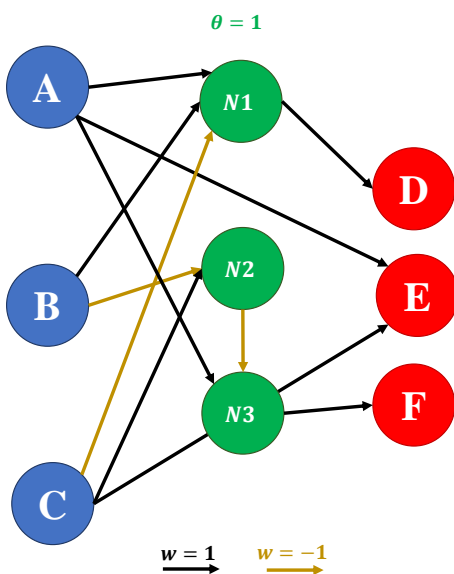
شکل ۷: شبکه‌ای که می‌تواند برای خروجی F در نظر گرفته شود (دارای ایراد جزئی. درستش در شکل ۸).



شکل ۸: شبکه‌ای که می‌تواند برای خروجی F در نظر گرفته شود.



شکل ۹: شبکه ادغام شده (دارای ایراد جزئی. درستش در شکل ۱۰).



شکل ۱۰: شبکه ادغام شده بهینه (حالت ایده آل با ملاحظه).

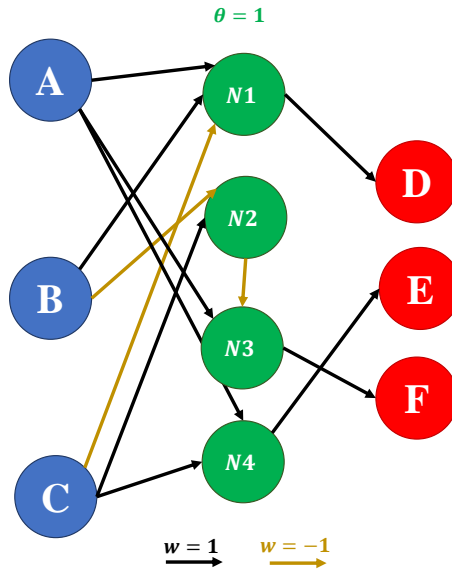
#### ۴.۱ پاسخ قسمت د

برای این قسمت از سوال، ضمن انجام یکسری بهینه سازی نهایی روی شبکه دستورات زیر را می نویسیم:

```

1 class McCullochPittsNeuron:
2     def __init__(self, weights, threshold, bias=0):
3         self.weights = weights
4         self.bias = bias
5         self.threshold = threshold

```



شکل ۱۱: شبکه ادغام‌شده بهینه (حالتی دیگر).

```

6
7     def fire(self, inputs):
8         weighted_sum = sum(w * x for w, x in zip(self.weights, inputs)) + self.bias
9         if weighted_sum >= self.threshold:
10             return 1
11         else:
12             return 0
13
14 class ThreeInputThreeOutputNetwork:
15     def __init__(self):
16         self.neuron_D = McCullochPittsNeuron([1, 1, -1], 1)
17         self.neuron_F1 = McCullochPittsNeuron([1, -1], 1)
18         self.neuron_F = McCullochPittsNeuron([1, 1], 1)
19
20     def predict(self, inputs):
21         A, B, C = inputs
22         D = self.neuron_D.fire([A, B, C])
23         E_inputs = [A, C]
24         E = self.neuron_D.fire(E_inputs)
25         F1 = self.neuron_F1.fire([B, C])
26         F_inputs = [A, F1]
27         F = self.neuron_F.fire(F_inputs)
28         return D, E, F
29
30 # Test the network for all possible input states

```

```

31 network = ThreeInputThreeOutputNetwork()
32
33 for A in range(2):
34     for B in range(2):
35         for C in range(2):
36             D, E, F = network.predict([A, B, C])
37             print(f"Input: ({A}, {B}, {C}) --> Output: ({D}, {E}, {F})")

```

انواع ترکیب‌ها و روش‌های دیگری در کدهای پیاده‌سازی آورده شده در این لینک آزمایش شده است که براساس روش خواسته شده در صورت سوال، روش کدنویسی‌ای که به صورت صریح از گیت‌های منطقی استفاده نکرده است و کاملاً نورون پایه است صحیح است. با اجرای دستورات، نتایج به شرح زیر خواهد بود و مشاهده می‌شود که خواسته سوال کاملاً برآورده شده است:

```

1 Input: (0, 0, 0) --> Output: (0, 0, 0)
2 Input: (0, 0, 1) --> Output: (0, 1, 0)
3 Input: (0, 1, 0) --> Output: (1, 0, 0)
4 Input: (0, 1, 1) --> Output: (0, 1, 0)
5 Input: (1, 0, 0) --> Output: (1, 1, 1)
6 Input: (1, 0, 1) --> Output: (0, 1, 0)
7 Input: (1, 1, 0) --> Output: (1, 1, 1)
8 Input: (1, 1, 1) --> Output: (1, 1, 1)

```

هم‌چنین دستورات زیر را نوشته‌ایم که بر اساس شبکه طراحی شده و منطق گفته شده و با دریافت حالت کنونی و ورودی، مراحل پذیرش با رویت ۱۰۰ را نشان دهد:

```

1 # Test the network with initial state A=0, B=0
2 network = ThreeInputThreeOutputNetwork()
3
4 # Define the initial state
5 A, B = 0, 0
6
7 while True:
8     # Get the input C
9     C = int(input("Enter input C (0 or 1): "))
10
11     # Get the next state and acceptance
12     D, E, F = network.predict([A, B, C])
13
14     # Print the output
15     print(f"Current state: ({A}, {B}), Input: {C} --> Next state: ({D}, {E}), Acceptance: {F}")
16
17     # Update the current state
18     A, B = D, E
19
20     # Check if the acceptance is 1, and break the loop if it is
21     if F == 1:
22         print("Acceptance reached!")

```

23      `break`

نتیجه یک نمونه از پیاده‌سازی به صورت زیر است که مشاهده می‌شود کاملاً منطبق با خواست سوال است:

```

1 Enter input C (0 or 1): 0
2 Current state: (0, 0), Input: 0 --> Next state: (0, 0), Acceptance: 0
3 Enter input C (0 or 1): 1
4 Current state: (0, 0), Input: 1 --> Next state: (0, 1), Acceptance: 0
5 Enter input C (0 or 1): 1
6 Current state: (0, 1), Input: 1 --> Next state: (0, 1), Acceptance: 0
7 Enter input C (0 or 1): 0
8 Current state: (0, 1), Input: 0 --> Next state: (1, 0), Acceptance: 0
9 Enter input C (0 or 1): 0
10 Current state: (1, 0), Input: 0 --> Next state: (1, 1), Acceptance: 1
11 Acceptance reached!

```

این دستور را به یک صورت دیگر نیز پیاده‌سازی کرده‌ایم:

```

1 # Take input from user
2 current_state = [int(x) for x in input("Enter current state A,B (separated by comma): ").split(',')]
3 C = int(input("Enter input C: "))
4 next_state = current_state.copy()
5
6 # Test the network for the given input
7 network = ThreeInputThreeOutputNetwork()
8
9 while True:
10     D, E, F = network.predict([current_state[0], current_state[1], C])
11     next_state[0], next_state[1] = D, E
12     if F == 1:
13         break
14     current_state = next_state.copy()
15     C = int(input("Enter input C: "))
16
17 # Print the final output
18 print(f"Current State: ({current_state[0]}, {current_state[1]}) --> Input: {C} --> Next State: ({next_state[0]}, {next_state[1]}) --> Acceptance: {F}")
19
20
21 # Take input from user
22 current_state = [int(x) for x in input("Enter current state A,B (separated by comma): ").split(',')]
23 C = int(input("Enter input C: "))
24 next_state = current_state.copy()
25
26 # Test the network for the given input
27 network = ThreeInputThreeOutputNetwork()

```

```

28
29 # Flag variable to track if F=1 has occurred previously or not
30 f_occurred = False
31
32 while True:
33     D, E, F = network.predict([current_state[0], current_state[1], C])
34     next_state[0], next_state[1] = D, E
35     if F == 1 and not f_occurred:
36         f_occurred = True
37     else:
38         current_state = next_state.copy()
39         C = int(input("Enter input C: "))
40
41     # Print output only if F=1 has occurred previously or F is not equal to 1
42     if f_occurred or F != 1:
43         print(f"Current State: ({current_state[0]}, {current_state[1]}) --> Input: {C} --> Next
44             State: ({next_state[0]}, {next_state[1]}) --> Acceptance: {F}")
45
46     if f_occurred:
47         break
48
49 # Print the final output
50 print(f"Current State: ({current_state[0]}, {current_state[1]}) --> Input: {C} --> Next State: ({
51     next_state[0]}, {next_state[1]}) --> Acceptance: {F}")

```

نتیجه دو نمونه از پیاده‌سازی با استفاده از روش دیگر به صورت زیر است که با منطق خواسته شده سازگار است:

```

1 Enter current state A,B (separated by comma): 0,0
2 Enter input C: 0
3 Enter input C: 1
4 Enter input C: 1
5 Enter input C: 0
6 Enter input C: 0
7 Current State: (1, 0) --> Input: 0 --> Next State: (1, 1) --> Acceptance: 1
8 -----
9
10 Enter current state A,B (separated by comma): 1,1
11 Enter input C: 0
12 Current State: (1, 1) --> Input: 0 --> Next State: (1, 1) --> Acceptance: 1
13 Current State: (1, 1) --> Input: 0 --> Next State: (1, 1) --> Acceptance: 1

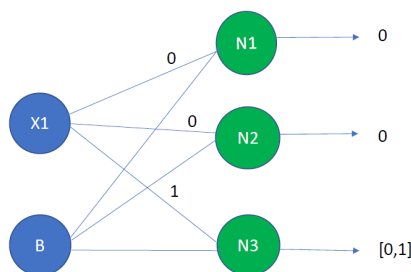
```

## ۵.۱ راه حل و نگاه دوم

ما برای حل این چالش دو منطق متفاوت را پیش بردیم که در ادامه با شرح مختصری راجب این ماشین دیدگاه خود را مطرح خواهیم کرد. با توجه به سوال مطرح شده که در آن شبیه سازی یک ماشین متناهی قطعی خواسته شده است ما به مطالعه و

شناخت این رفتار این ماشین پرداختیم و با توجه به درکی که از این ماشین به دست آمد تلاش کردیم که در نخست رفتار هر یک از حالت‌های این ماشین را به صورت مجزا پیاده سازی کنیم که برای این کار از نرون Mcculloch-Pitts توسعه یافته بهره بردیم. در این سوال DFA به این صورت طراحی شده تا رشته 100 را دی یک دنباله از اعداد باینری را تشخیص دهد که در ادامه با توجه به ورودی‌های ممکن حرکات هر حالت از این ماشین را نشان می‌دهیم. در بهره‌گیری از این ماشین رشته ورودی ما در ابتدا بر روی یک ابزار ذخیره سازی ذخیره می‌شود و با استفاده از یک حد خواندن از یک سمت به صورت بیت به بیت خوانده شده و به حالتی که در آن قرار داریم اعمال می‌شود و این ماشین با توجه به رفتاری که برای آن تعریف شده و ورودی اعمال شده تصمیم‌گیری می‌کند. ما سه حالت مدنظر را با اعداد باینری شماره گذاری کرده‌ایم به این صورت که حالت اول (00)، حالت دوم (01) و حالت سوم (10).

### حالت اول



شکل ۱۲: تصویر مربوط به حالت اول در نگاه دوم.

توابع حاصل شده از نرون‌ها برای شبکه اول

$$N1 = X1.W1 + B = 0$$

$$N2 = X1.W2 + B = 0$$

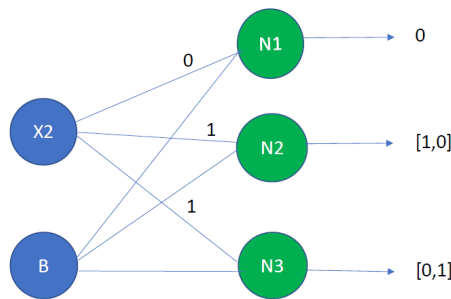
$$N3 = \begin{cases} 1 & \text{if } X1 = 1 \Rightarrow X1.W3 + B \\ 0 & \text{else} \end{cases}$$

در این حالت به ازای هر ورودی صفر و یک دو حرکت وجود دارد که البته حالت پذیرش و یا عدم پذیرش آن هم مدنظر می‌باشد. همانطور که گفته شد رشته 100 با یک شروع می‌شود پس تا لحظه دیدن یک در حالت اول می‌مانیم و تنها با دیدن یک به حالت بعدی می‌رویم (01). برای شبیه سازی این عملیات ما سه نرون در خروجی در نظر گرفته‌ایم که خروجی نرون  $N1$  حالت پذیرش و یا عدم پذیرش را مشخص می‌کند به این صورت که خروجی یک به معنای پذیرش و صفر به معنای عدم پذیرش می‌باشد و دو نرون  $N2$  و  $N3$  هر کدام یک بیت از شماره حات بعد را نمایش می‌دهند. لازم به ذکر است که مقدار  $bias$  و



*threshold*، صفر می باشد و همچنین روش مقایسه نوروں ما  $NET == 0$  می باشد. ما می دانیم که در حالت اول در هر صورت ما با عدم پذیرش روبه رو هستیم به این دلیل که در ابتدا تنها یک بیت از رشته مد نظر ما دیده شده است پس در هر صورت خروجی نوروں اول ما باید صفر باشد پس وزن ها و بایاس را جوری در نظر گرفتیم که خروجی نوروں اول صفر شود. نوروں های  $(N3, N2)$  قرار است با خروجی که تولید می کنند شماره حالت بعدی را نشان دهند. پس اگر بیت اول ورودی ما صفر بود که ما قرار است در همان حالت فعلی بمانیم (00) این به این معناست که نوروں های  $(N2, N3)$  باید صفر را تولید کنند. اما اگر بیت اول ورودی ما یک بود ما باید به حالت (01) برویم که در این صورت نوروں  $N2$  صفر می ماند و نوروں  $N3$  که بیت کم ارزش را برای آن در نظر گرفته ایم یک می شود که در جدول و شکل نوروں ها همه ی حالت های ممکن درج شده است.

### حالت دوم



شکل ۱۳: تصویر مربوط به حالت دوم در نگاه دوم.

### توابع حاصل از نوروں ها برای شبکه دوم

$$N1 = X2.W1 + B = 0$$

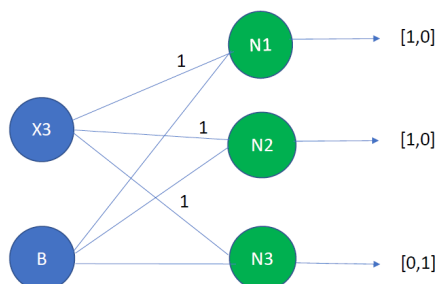
$$N2 = \begin{cases} 1 & \text{if } X2 = 0 \Rightarrow X2.W2 + B \\ 0 & \text{else} \end{cases}$$

$$N3 = \begin{cases} 1 & \text{if } X2 = 1 \Rightarrow X2.W3 + B \\ 0 & \text{else} \end{cases}$$

توضیحات نوروں  $N1$  دقیقاً مشابه حالت قبل است به این معنی که چون هنوز دنباله مورد نظر ما قطعاً در این مرحله نمی تواند دیده شده باشد پس باید خروجی آن صفر در نظر گرفته شود. ما باید خود را کاملاً در این حالت متصور باشیم به این معنی که به صورت منطقی ما اینگونه به این موضوع نگاه می کنیم که حتماً یک را دیده ایم که اکنون در این حالت قرار داریم پس در صورت دیدن صفر که بیت دوم مورد نظر ما بعد از یک است باید به حالت بعد یعنی حالت سوم (10) برویم پس با توجه به این نکات اگر در این حالت صفر را دیدیم نوروں  $N2$  باید یک شود و نوروں  $N3$  باید صفر شود که نشان گر حالت سوم می باشد اما اگر ورودی

ما یک بود مه باید همچنان در حالت کنونی که (01) می باشد بمانیم که در این صورت نورون  $N2$  صفر و نورون  $N3$  باید یک شود که نمایان گر همان حالت فعلی است. (01). لازم به ذکر است که این شبکه دارای  $bias$  و  $threshold$  یک است و روش مقایسه آن برای هر نورون متفاوت است.

### حالت سوم



شکل ۱۴: تصویر مربوط به حالت سوم در نگاه دوم.

### توابع حاصل از نورون ها برای شبکه سوم

$$N1 = \begin{cases} 1 & \text{if } X3 = 0 \Rightarrow X3.W1 + B \\ 0 & \text{else} \end{cases}$$

$$N2 = \begin{cases} 1 & \text{if } X3 = 0 \Rightarrow X3.W2 + B \\ 0 & \text{else} \end{cases}$$

$$\text{for } X3 = 0, 1 \Rightarrow X3.W3 + B = 1$$

نورون  $N1$  در صورت دیده شدن صفر حالت پذیرش را یک می کند به این دلیل که با توجه به در این حالت قرار گرفتن می دانیم که حتما تا اینجای کار رشته (10) دیده شده است پس بادیکن یک صفر دیگر رشته مد نظر ما کاملا دیده شده. به همین ترتیب نورون های  $N2$ ,  $N3$  باید با دیدن صفر دیگر حالت نهایی را نمایش دهند که (11) می باشد پس باید هردو یک را تولید کنند و اما اگر بیت یک را دیدند هم حالت پذیرش باید صفر شود و هم حالت قبل را باید در خروجی نمایش دهند که در جدول دقیقا ذکر شده است. همچنین  $bias$  و  $threshold$  یک در نظر گرفته شده است.

N3	N2	N1	ورودی
۱	۱	۱	۰
۱	۰	۰	۱

حالت سوم:

N3	N2	N1	ورودی
۰	۱	۰	۰
۱	۰	۰	۱

حالت دوم:

N3	N2	N1	ورودی
۰	۰	۰	۰
۱	۰	۰	۱

حالت اول:

حالت اول			
ورودی	N1	N2	N3
0	0	0	0
1	0	0	1
حالت دوم			
ورودی	N1	N2	N3
0	0	1	0
1	0	0	1
حالت سوم			
ورودی	N1	N2	N3
0	1	1	1
1	0	0	1