تمرین اول درس معماری کامپیوتر

دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف

آرین احدی نیا

استاد درس: جناب آقای دکتر جهانگیر

دستیار آموزشی: جناب آقای علی‌پور

پاییز ۱۴۰۰

فهرست عناوین

مفاهیم اولیه

سوال ۱

توجه کنید که مدارها و توابع ترکیبی، در پیچیده‌ترین حالت یک بردار از ورودی‌های منطقی را دریافت و یک بردار از خروجی‌های منطقی را برمیگردانند؛ یعنی که چند ورودی و چند خروجی دارند. در حالت ساده‌تر، یک تابع ترکیبی یک بردار منطقی دریافت و به عنوان خروجی یک تک مقدار منطقی برمی‌گرداند؛ یعنی تابع چند ورودی و یک خروجی دارد.

یک مدار با یک چند خروجی را می‌توان به صورت چند مدار با ورودی‌های مشابه ولی با دقیقا یک خروجی در نظر گرفت که کنار یکدیگر قرار گرفته‌اند. بنابرین اگر بتوانیم یک مدار تک خروجی را با استفاده از Mux بسازیم، با قرار دادن چند مدار در کنار هم می‌توانیم مدارهای چند خروجی را نیز پیاده‌سازی کنیم.

یک مدار با n خط ورودی، می‌تواند 2^n حالت ترکیب مختلف در ورودی ایجاد کند. برای ساخت این مدار به وسیله Mux میتوانیم ورودی‌ها را به خطوط انتخاب $(Select)$ بدهیم و خروجی متناظر هر ترکیب از ورودی را، به خط ورودی متناظر آن ترکیب در ورودی I بدهیم. توجه کنید که خروجی متناظر هر یک از ترکیب‌ها برابر مقدار منطقی صفر و یا یک است.

به بیان دقیق‌تر اگر Mux مورد نظر دارای n خط انتخابی S_0، S_1، S_2، ... و S_{n-1} و 2^n خط ورودی I_0، I_1، I_2، ... و I_{2^n-1} باشد. برای ساخت تابع ترکیبی دلخواه f با n ورودی A_0، A_1، A_2، ... A_{n-1} به طریق زیر عمل می‌کنیم.

- ورودی $A_{n-1}A_{n-2} \dots A_1A_0$ را به خط‌های انتخابی $S_{n-1}S_{n-2} \dots S_1S_0$ در Mux متصل می‌کنیم.
- به ازای هر $0 \le j < 2^n - 1$، مقدار $f(j)$ را که صفر یا یک است به خط ورودی I_j متصل می‌کنیم. (توجه کنید که j در نمایش باینری، یک عدد n بیتی است و شامل n بیت ورودی تابع f می‌شود.)

زمانی که ورودی به مدار وارد میشود، خط متناظر آن ورودی در ورودی Mux به خروجی متصل می‌گردد که همان جواب متناظر با آن ورودی است.

پایه نظری این روش بر مبنای بسط شنون است. بسط شنون بیان میدارد که اگر f یک تابع منطقی از b_0، b_1، b_2 الی b_n باشد، خواهیم داشت

$$f(b_0, b_1, b_2, \dots, b_n) = b_0' f(0, b_1, b_2, \dots, b_n) + b_0 f(1, b_1, b_2, \dots, b_n)$$

بنابرین اگر $n+1$ مرحله بسط شنون را بر روی تابع f اعمال کنیم خواهیم داشت.

$$f(b_0, b_1, b_2, \dots, b_n) = b_0' f(0, b_1, b_2, \dots, b_n) + b_0 f(1, b_1, b_2, \dots, b_n)$$

$$= b_0' b_1' f(0,0,b_2,b_3,\dots,b_n) + b_0' b_1 f(0,1,b_2,b_3,\dots,b_n) + b_0 b_1' f(1,0,b_2,b_3,\dots,b_n) + b_0 b_1 f(1,1,b_2,b_3,\dots,b_n)$$

$$= b_0' b_1' b_2' f(0,0,0,b_3,\dots,b_n) + b_0' b_1' b_2 f(0,0,1,b_3,\dots,b_n) + b_0' b_1 b_2' f(0,1,0,b_3,\dots,b_n) + b_0' b_1 b_2 f(0,1,1,b_3,\dots,b_n) + b_0 b_1' b_2' f(1,0,0,b_3,\dots,b_n) + b_0 b_1' b_2 f(1,0,1,b_3,\dots,b_n) + b_0 b_1 b_2' f(1,1,0,b_3,\dots,b_n) + b_0 b_1 b_2 f(1,1,1,b_3,\dots,b_n)$$

$$= \cdots = m_0 f(0) + m_1 f(1) + m_2 f(2) + \cdots + m_{2^{n+1}-1} f(2^{n+1}-1)$$

توجه بفرمایید که در مرحله آخر m_i برابر $min - term$ها هستند و ورودی تابع f یک عدد $n+1$ بیتی است.

تابع فوق را می‌توانیم با استفاده از MUX بسازیم. **توجه کنید که $f(i)$ها در مرحله آخر مقدار صریح صفر یا یک را دارند چرا که ورودی آنها تماما مشخص است**. بدون از دست دادن کلیت فرض کنید که $b_0 b_1 b_2 \dots b_n = i$ باشد. در این صورت $m_i = 1$ خواهد بود و به ازای هر $j \neq i$، $m_j = 0$ خواهد بود. بنابرین خواهیم داشت

$$f(b_0,b_1,b_2,\dots,b_n) = m_0 f(0) + \cdots + m_i f(i) + \cdots + m_{2^{n+1}-1} f(2^{n+1}-1)$$

$$= 0.f(0) + 0.f(1) + \cdots + 1.f(i) + \cdots + 0.f(2^{n+1}-2) + 0.f(2^{n+1}-1)$$

$$= f(i)$$

بنابرین کافی است که به ورودی متناظر ترکیب انتخابی i در MUX، مقدار صریح $f(i)$ را وصل کنیم.

توجه کنید که بر مبنای همین روابط، یک تابع با n ورودی و یک خروجی را میتوان تنها با استفاده از $2^n - 1$ مولتی‌پلکسر ۲ به ۱ پیاده‌سازی کرد. این امر را میتوان به طور دقیق به وسیله استقرا ضعیف اثبات کرد.

استقرا:

پایه: یک تابع ترکیبی با یک ورودی و دو خروجی را میتوان به کمک یک مولتی‌پلکسر ۲ به ۱ ساخت.

اثبات: کافی است ورودی تابع $f(b_0)$ را به یک مولتی‌پلکسر ۲ به ۱ وصل کنیم و مقدار $f(0)$ را به خط ورودی I_0 و ورودی $f(1)$ را به ورودی I_1 وصل کنیم. بنابرین تابع مورد نظر با $2^1 - 1 = 1$ مولتی‌پلکسر با این نحو ساخته می‌شود.

گام: اگر بتوانیم یک تابع با n ورودی را به کمک $2^n - 1$ مولتی‌پلکسر های ۲ به ۱ پیاده‌سازی کنیم، یک تابع با $n+1$ ورودی را نیز می‌توانیم با استفاده از $2^{n+1} - 1$ مولتی‌پلکسر ۲ به ۱ پیاده‌سازی کنیم.

اثبات: فرض کنید تابع مورد نظر $f(b_0,b_1,b_2,\dots,b_n)$ باشد. طبق بسط *Shannon* داریم

$$f(b_0, b_1, b_2, \dots, b_n) = b_0' f(0, b_1, b_2, \dots, b_n) + b_0 f(1, b_1, b_2, \dots, b_n)$$

توجه فرمایید که توابع $f(1, b_1, b_2, \dots, b_n)$ و $f(0, b_1, b_2, \dots, b_n)$ دو تابع با n ورودی هستند. طبق فرض استقرا، می‌توانیم هر یک از آنها را به وسیله $2^n - 1$ مولتی‌پلکسر ۲ به ۱ بسازیم.

برای ساخت تابع f میتوانیم که ورودی b_0 را به ورودی یک مولتی‌پلکسر ۲ به ۱ متصل کنیم. سپس تابع $f(0, b_1, b_2, \dots, b_n)$ را به خط I_0 و تابع $f(1, b_1, b_2, \dots, b_n)$ را به خط I_1 متصل کنیم. مجموعا در روش فوق از

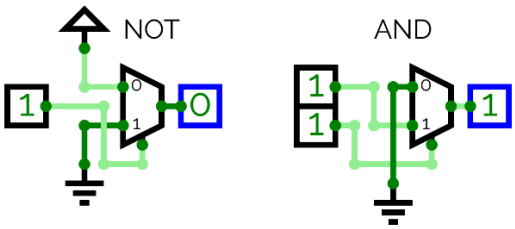
$$(2^n - 1) + (2^n - 1) + 1 = 2 \times 2^n - 1 = 2^{n+1} - 1$$

مولتی‌پلکسر ۲ به یک استفاده می‌شود.

بنابرین به وسیله استقرا اثبات کردیم که یک تابع دلخواه ترکیبی با n ورودی را می‌توان به وسیله $2^n - 1$ مولتی‌پلکسر ۲ به ۱ پیاده سازی کرد. با توجه به توصیفی که پیشتر از توابع ترکیبی با چند خروجی کردیم، میتوانیم یک تابع با m خروجی را معادل m تابع با ورودی‌های مشابه و یک خروجی در نظر گرفت. بنابرین یک تابع ترکیبی به n ورودی و m خروجی را میتوان به کمک $m(2^n - 1)$ مولتی‌پلکسر ۲ به ۱ پیاده‌سازی کرد.

با توجه به اینکه هر تابع ترکیبی از مقادیر منطقی را می‌توانیم به وسیله MUX و مقادیر منطقی صفر و یک بسازیم، اگر به مولتی‌پلکسر به چشم یک گیت منطقی با سه ورودی و یک خروجی نگاه کنیم، مجموعه شامل گیت MUX، به همراه منبع فعال و غیر فعال سازنده یک منطق کامل هستند. با توجه به اینکه به منابع فعال و غیر فعال برای ساخت این مدارات نیاز است، گیت MUX یک منطق کامل ضعیف است. (بر خلاف $\{NAND\}$ و یا $\{NOR\}$ که بدون نیاز به منابع فعال و غیر فعال سازنده یک منطق کامل هستند.)

روش دیگری که میتوانستیم برای حل این مساله به‌کار ببندیم این بود که نشان دهیم مجموعه مذکور سازنده یک منطق کامل است. برای این منظور کافی است نشان دهیم که اپراتور های یک مجموعه منطق کامل به وسیله این منطق قابل ساخت هستند. به عنوان مثال، میدانیم که مجموعه $\{AND, NOT\}$ سازنده یک منطق کامل است. به طریق زیر میتوانیم این دو اپراتور را به وسیله گیت MUX و منابع فعال و غیر فعال تولید کنیم. بنابرین هر تابعی که به وسیله منطق $\{AND, NOT\}$ قابل تولید باشد، با منطق $\{MUX, 0, 1\}$ نیز قابل تولید خواهد بود. بنابرین منطق $\{MUX, 0, 1\}$ نیز کامل خواهد بود. بدیهی است که با ساخت مدار به این نحو، بهینه تر از حالت قبل خواهد بود.



سوال ۲

در یک رمزگشا، n ورودی و 2^n خروجی وجود دارد. به ازای هر ترکیبی از n ورودی، دقیقا یک خط متناظر در خروجی فعال میگردد. این تناظر به این صورت است که شماره خط خروجی به شکل یک عدد n بیتی مانند i در مبنای دو بین 0 و $2^n - 1$ در ورودی به رمزگشا داده میشود و در خروجی خط iاُم فعال می‌گردد.

رمزگذار وارون رمزگشا است. یعنی 2^n خط ورودی و n خط خروجی دارد. زمانی که در ورودی یک خط فعال شود، شماره آن خط در خروجی مشخص می‌گردد.

مشکلی که در رمزگذار می‌تواند به وجود بیاید این است که به جای یک خط، چند خط در ورودی فعال باشند. توجه کنید که از آنجایی تنها یک خروجی برای رمزگذار قابل نمایش است، رمزگذار تنها شماره یکی از خطوط فعال شده را میتواند در خروجی نمایان کند.

راهکار مرسومی که برای حل این مشکل وجود دارد. استفاده از رمزگذار **الویت‌دار** است. رمزگذار الویت‌دار به این صورت است که هر یک از خطوط ورودی دارای یک الویت هستند که معمولا به ترتیب صعودی یا نزولی بر حسب شماره خط است. زمانی که چند خط در ورودی همزمان با هم فعال شوند، شماره خط با الویت بیشتر در خروجی نمایان می‌گردد.

مشکل دیگری که در رمزگذار میتواند به وجود بیاید این است که به جای اینکه در ورودی یک یا چند خط فعال باشند، هیچ خطی فعال نباشد. در این حالت، خروجی $don't\ care$ خواهد بود اما هیچ راه تمایزی میان آنکه این خروجی به علت نبود خط فعال در ورودی نمایش داده شده و یا واقعا آن خط در ورودی فعال بوده است، وجود ندارد. راهکاری که برای حل این مشکل وجود دارد، اضافه کردن یک خروجی $active$ است. این خروجی در زمانی که حداقل یک خط در ورودی فعال باشد، فعال میشود و در صورتی که هیچ خطی در ورودی فعال نباشد، غیرفعال میشود. به سادگی این خروجی را با OR کردن تمام خطوط ورودی می‌توانیم تولید کنیم.

سوال ۳

$DeMux$ به این صورت که ورودی $Select$ یک عدد باینری تلقی می‌شود و خط با آن شماره در تناظر با آن ترکیب ورودی قرار می‌گیرد. زمانی که ورودی $Select$ یک خط خروجی را مشخص می‌کند، ورودی D به آن خط متصل میشود و بقیه خطوط غیر فعال میشوند. به عبارت دیگر اگر ورودی D فعال باشد، در خروجی تنها خط متناظر با ورودی $Select$ فعال میشود و در غیر این صورت تمامی خطوط غیرفعال می‌شوند.

رمزگشا با ورودی $Enable$ به این صورت است که n خط ورودی I و یک ورودی $Enable$ و 2^n خط خروجی O دارد. در صورتی که ورودی $Enable$ فعال باشد، به ازای هر ترکیب از ورودی دقیقا یک خط متناظر در خروجی فعال میشود. در صورتی که $Enable$ فعال نباشد، تمامی خطوط غیرفعال می‌شوند.

ورودی‌ها و خروجی‌های $DeMux$ دقیقا در تناظر با یک دیگر قرار دارند.

تعداد بیت	متناظر در رمزگشا	متناظر در $DeMux$
۱	$Enable$	D
n	I	$Select$
2^n	O	Y

بنابرین تنها با تغییر نام ورودی‌ها، می‌توان یک $DeMux$ را به یک رمزگشا با ورودی $Enable$ تبدیل کرد و برعکس.

در صورتی که ورودی $Enable$ در رمزگشا مورد نیاز نباشد، میتوانیم ورودی D را در $DeMux$ به منبع همواره فعال متصل کنیم. توجه کنید که رمزگشا بدون ورودی $Enable$ را نمی‌توان به یک $DeMux$ تبدیل کرد.

سوال ۴

توجه کنید که مجموعه شامل اپراتور $NAND$ سازنده یک منطق کامل است. به این معنا که با استفاده از این اپراتور، تمام توابع منطقی قابل ساخت هستند. برای اثبات این موضوع، کافیست نشان دهیم که اپراتور AND و OR و NOT به وسیله این منطق قابل ساخت هستند. از آنجایی که هر تابع منطقی به صورت SOP و یا POS به وسیله این سه اپراتور قابل ساخت است، نتیجه خواهیم گرفت که آن تابع به وسیله اپراتور $NAND$ نیز قابل ساخت خواهد بود.

$$NOT(A) = NAND(A, A)$$

$$AND(A,B) = NOT(NAND(A,B) = NAND\big(NAND(A,B), NAND(A,B)\big)$$

$$OR(A,B) = NAND\big(NOT(A), NOT(B)\big) = NAND\big(NAND(A,A), NAND(B,B)\big)$$

بنابرین با جایگزینی تمام گیت‌های در مدارهای SOP و یا POS می‌توان آنها تماما با گیت‌های $NAND$ ایجاد کرد. مشکلی که در این روش وجود دارد این است که مدار از حالت بهینه از منظر سرعت خارج می‌شود و مدار بیش از دو طبقه میشود. بالاخص که ساخت گیت‌های AND و OR هر کدام به تنهایی شامل دو طبقه می‌شوند. بنابرین باید به دنبال راهی بگردیم که مانند ساده‌سازی POS و یا SOP، بتوانیم مدار را با دو طبقه بسازیم.

می‌خواهیم نشان دهیم که با استفاده از فرآیند ساده‌سازی SOP، می‌توانیم مدار را با استفاده از گیت‌های $NAND$ به صورت دو طبقه بسازیم.

فرض کنید یک تابع ترکیبی دلخواه به صورت زیر به شکل SOP ساده‌سازی شده است. توجه کنید که P_i ها هر یک برداری از چند مقدار منطقی هستند که به عنوان ورودی عناصر آنها به اپراتور ها داده میشود. با توجه به خاصیت جابجایی پذیری اپراتورهای مورد بحث، ترتیب عناصر در آن اهمیتی ندارد.

$$f(P) = OR\big(AND(P_l), AND(P_2), AND(P_3), \dots, AND(P_k)\big)$$

با توجه به قانون دمورگان داریم:

$$f(P) = \Big[\big[OR\big(AND(P_l), AND(P_2), AND(P_3), \dots, AND(P_k)\big)\big]'\Big]'$$

$$= [AND(AND(P_1)', AND(P_2)', AND(P_3)', \dots, AND(P_k)')]'$$

$$= NAND\big(NAND(P_1), NAND(P_2), NAND(P_3), \dots, NAND(P_k)\big)$$

بنابرین اگر در ساده‌سازی SOP تمام گیت‌ها را با $NAND$ جایگزین کنیم. مدار عملکرد خود را حفظ خواهد کرد. بنابرین برای ساخت مدار فوق با استفاده از گیت های $NAND$ کافی است که تابع مورد نظر مشابه حالت SOP ساده‌سازی کنیم و در نهایت تمام گیت‌ها را با $NAND$ جایگزین کنیم. توجه کنید که بنابر استدلال مشابه SOP و POS، برای ساخت توابع ترکیبی اگر ورودی‌ها و نقیض آنها را داشته باشیم، به حداقل دو طبقه نیاز است. بنابرین این روش از نظر تعداد طبقه بهینه است.

de \ bc	00	01	11	10
00	**0**	**0**	**1**	**1**
01	**0**	**0**	**0**	***X***
11	**0**	***X***	***X***	**1**
10	***X***	**0**	**1**	**1**

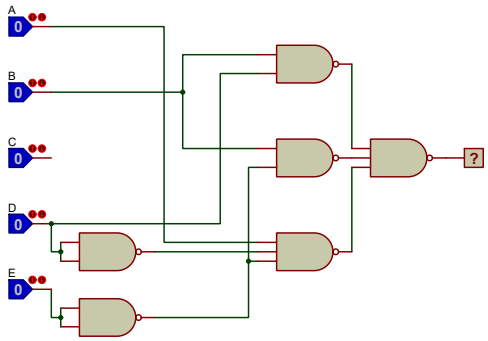
$a = 0$

de \ bc	00	01	11	10
00	**1**	***X***	**1**	**1**
01	**0**	**0**	***X***	**0**
11	**0**	**0**	**1**	**1**
10	**0**	**0**	**1**	**1**

$a = 1$

$$f(a,b,c,d,e) = bd + be' + ad'e' = \big((bd)'(be')(ad'e')\big)'$$

بنابرین مدار مورد نظر به شکل زیر قابل ساخت است.



توجه بفرمایید که دو گیت در کنار ورودی D و E برای تولید نقیض این ورودی‌ها هستند. در شمارش گیت‌ها فرض می‌شود که ورودی‌ها و نقیض آنها آماده وجود دارند.

بنابرین مدار مورد نظر با استفاده از دو گیت سه ورودی و دو گیت دو ورودی قابل ساخت است.

سوال ۵

یک حافظه ROM با عرض بیتی ۱ به این صورت است که η خط آدرس‌دهی و یک خط خروجی یک بیتی دارد. متناظر با هر یک از ترکیب‌های خطوط آدرس‌دهی، یک مقدار یک بیتی گویی که در حافظه ذخیره شده است و با وارد شدن آن ترکیب در ورودی، مقدار ذخیره شده در خروجی ظاهر می‌گردد. حافظه ROM با عرض بیتی ω نیز مشابه است با این تفاوت که خروجی و مقادیر ذخیره شده به جای یک بیت، ω بیت دارند.

یک مدار ترکیبی دلخواه با n ورودی و m خروجی را در نظر بگیرید. متناظر با هر ترکیب از این n بیت ورودی، یک خروجی m بیتی مشخص ظاهر می‌شود.

ارتباط این دو مدار بسیار واضح است. برای ساخت یک مدار ترکیبی با n ورودی و m خروجی، می‌توانیم از یک ROM با n خط آدرس‌دهی و عرض بیتی m استفاده کنیم. بدین منظور n ورودی تابع را به خطوط انتخابی ROM متصل می‌کنیم. سپس خروجی m بیتی متناظر با هر ترکیب از ورودی را در حافظه متناظر با آن ترکیب در ROM ثبت می‌کنیم. در این حالت زمانی که ترکیب ورودی وارد شود، خروجی متناظر ظاهر می‌گردد. عملکرد این مدار دقیقا متناظر تابع ترکیبی خواهد بود.

مبانی نظری ساخت مدار به این صورت دقیقا مشابه ساخت مدار با m عدد $MUX\ n{:}\ 1$ است که در سوال ۱ مطرح کردیم.

سوال ۶

در صورت استفاده از مدارهای برنامه‌پذیر مانند EP-ROM نحوه کلی ساخت مدار تغییر نخواهد کرد. اما میتوانیم با استفاده از روشی مدار را از ابتدا برنامه‌ریزی کنیم. به عنوان مثال در مورد EP-ROM می‌توانیم با تاباندن نور فرابنفش محتویات حافظه مدار را پاک و آن را از ابتدا برنامه‌ریزی کنیم. بنابرین می‌توانیم در صورت لزوم مدار را به نحوی بروزرسانی کنیم. همچنین برای کاربردهای موقت می‌توانیم پس از اتمام آن کاربرد از حافظه برای منظور دیگری استفاده کنیم.

در مقابل، هزینه استفاده از چنین مداری به قطع یقین بیشتر است. همچنین چنین مدار آسیب‌پذیر تر است. به عنوان مثال، تشعشع‌های کیهانی تاثیر بیشتری بر مدارهایی مانند EP-ROM خواهند داشت.

سوال ۷

همانگونه که در سوال ۵ گفتیم، برای ساخت یک مدار با n ورودی و m خروجی نیاز به یک حافظه ROM با n خط آدرس‌دهی و عرض بیتی m می‌باشیم.

الف. یک جمع و تفریق کننده ۳۲ بیتی، دو عدد ۳۲ بیتی به همراه یک ورودی کنترلی برای جمع یا تفریق به همراه یک ورودی C_{in} دریافت می‌کند. حاصل این عملیات یک عدد ۳۲ بیتی به همراه یک بیت C_{out} خواهد بود. بنابرین این مدار ۶۶ بیت ورودی و ۳۳ بیت خروجی دارد. بنابرین برای ساخت این مدار به یک حافظه ROM با ۶۶

خط ورودی و عرض بیتی ۳۳ نیازمندیم. بنابرین حافظه باید 2^{66} کلمه ۳۳ بیتی داشته باشد. بنابرین به یک حافظه 32×2^{66} بیتی نیازمندیم.

محاسبه محتوای خانه‌های این حافظه بدیهی است. فرمت آدرس‌دهی را به این صورت در نظر می‌گیریم.

۱	۳۲ بیت	۳۲ بیت	۱

که به ترتیب از سمت چپ ورودی‌های کنترلی $\overline{ADD}/SUB$، ورودی اول، ورودی دوم و بیت C_{in} می‌باشد. بنابرین به سادگی می‌توانیم مقدار ذخیره‌شده را مشخص کنیم. توجه کنید که خروجی C_{out} به نحوی پرارزش‌ترین بیت تلقی میگردد. بنابرین بیت اول ذخیره‌شده را C_{out} و مابقی را حاصل ۳۲ بیتی در نظر میگیریم.

$\overline{ADD}/SUB$	عدد اول	عدد دوم	C_{in}	بیت اول خروجی (C_{out})	حاصل ۳۲ بیتی
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	10
⋮	⋮	⋮	⋮	⋮	⋮
0	100101	10110	1	0	111100
⋮	⋮	⋮	⋮	⋮	⋮
1	0	0	0	0	0
1	0	0	1	0	1
1	0	1	0	0	111 ... 111 (۳۲ بار)
1	0	1	1	0	0
⋮	⋮	⋮	⋮	⋮	⋮
1	100101	10110	1	0	10000
⋮	⋮	⋮	⋮	⋮	⋮

توجه کنید که یک نوع ساده‌تر از مدار فوق این است که ورودی $\overline{ADD}/SUB$ و C_{in} را یکسان در نظر بگیریم. در این سوال از این روش استفاده نکرده‌ایم.

ب. یک ضرب‌کننده ۳۲ در ۳۲ بیتی، دو ورودی ۳۲ بیتی دریافت می‌کند. بنابرین این مدار در مجموع ۶۴ بیت ورودی دارد. همچنین توجه کنید که ضرب دو عدد ۳۲ بیتی، حداکثر ۶۴ بیتی خواهد بود. بنابرین برای ساخت این

مدار به یک حافظه ROM با ۶۴ خط ورودی و عرض بیتی ۶۴ نیازمندیم. بنابرین حافظه باید 2^{64} کلمه ۶۴ بیتی داشته باشد. بنابرین به یک حافظه 64×2^{64} بیتی نیازمندیم.

محاسبه محتوای خانه‌های این حافظه بسیار بدیهی و سر راست است. اگر آدرس هر خانه را یک عدد ۶۴ بیتی در نظر بگیریم، محتوای هر خانه ضرب ۳۲ بیت اول آدرس در ۳۲ بیت دوم آدرس میشود.

۳۲ بیت اول آدرس (عدد اول)	۳۲ بیت دوم آدرس (عدد دوم)	خروجی
0	0	0
0	1	0
⋮	⋮	⋮
110	10	1100
110	11	10010
110	100	11000
⋮	⋮	⋮

ج. یک کدگذار الویت‌دار ۸ بیتی، در ورودی یک عدد ۸ بیتی را دریافت و در خروجی یک عدد سه بیتی را نمایش می‌دهد. همچنین ممکن است که خروجی $active$ را به همان صورت که در سوال ۲ توضیح دادیم بخواهیم به مدار اضافه کنیم (OR تمام ورودی‌ها). در این صورت مدار ۴ خروجی خواهد داشت. بنابرین برای ساخت این مدار به یک حافظه با ۸ خط ورودی و عرض بیتی ۴ نیاز خواهیم داشت. بنابرین حافظه باید 2^8 کلمه ۴ بیتی داشته باشد. بنابرین به یک حافظه 4×2^8 بیتی نیازمندیم.

نحوه مشخص کردن مقدار ذخیره شده برای خروجی نیز دقیقا مطابق جدول حالات کدگذار الویت دار است. خروجی ۴ بیتی را به این صورت در نظر میگیریم که ۳ بیت اول شماره خط فعال شده و بیت آخر خروجی $active$ باشد. در این مدار، الویت را از خط با شماره کمتر در نظر میگیریم.

آدرس ($I_7I_6I_5 \dots I_0$)	مقدار ذخیره شده
00000000	0000
00000001	0001
$0000001X$	0011
$000001XX$	0101
$00001XXX$	0111

$0001XXXX$	1001
$001XXXXX$	1011
$01XXXXXX$	1101
$1XXXXXXX$	1111

د. یک تسهیم‌کننده ۲ به ۱ چهار بیتی، دو ورودی چهار بیتی دارد. بنابرین به همراه دو ورودی $Select$ و $Enable$ مجموعا ۱۰ بیت ورودی خواهیم داشت. خروجی مدار نیز یک عدد چهار بیتی خواهد بود. بنابرین برای ساخت این مدار به یک حافظه با ۱۰ خط ورودی و عرض بیتی ۴ نیاز خواهیم داشت. بنابرین حافظه باید 2^{10} کلمه ۴ بیتی داشته باشد. بنابرین به یک حافظه 4×2^{10} بیتی نیازمندیم.

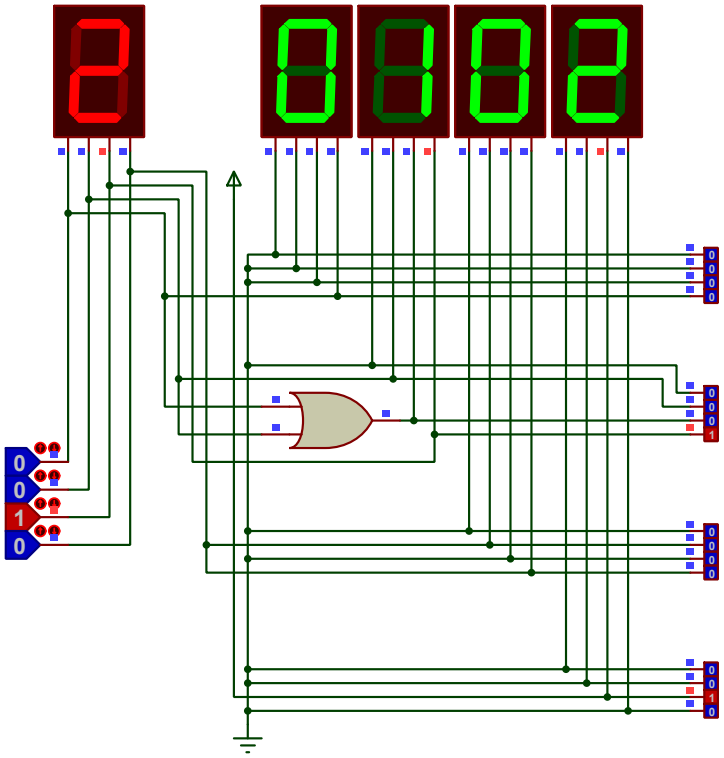
فرمت آدرس‌های ۱۰ بیتی را به این صورت در نظر میگیریم که ۴ بیت اول و دوم دو ورودی مولتی‌پلکسر و بیت ۹ام ورودی Sel و بیت ۱۰ام ورودی E باشد. خروجی نیز بنابر منطقی مولتی‌پلکسر، یکی از دو ورودی یا تمام صفر خواهد بود (در حالت غیر فعال). بنابرین برخی از آدرس‌ها و مقادیر ذخیره شده متناظر به شکل زیر خواهد بود.

عدد اول	عدد دوم	Sel	E	خروجی
$XXXX$	$XXXX$	X	0	0000
0000	0000	0	1	0000
0000	0000	1	1	0000
0000	0001	0	1	0000
0000	0001	1	1	0001
⋮	⋮	⋮	⋮	⋮
1001	0010	0	1	1001
1001	0010	1	1	0010
⋮	⋮	⋮	⋮	⋮

مدارهای ترکیبی

سوال ۱

با استفاده از تنها یک گیت OR و داشتن منابع فعال و زمین، میتوان مدار مورد نظر را به شکل زیر ساخت. توجه کنید که در شکل زیر، از نمایشگر های $7SEG\ BCD$ برای نمایش بهتر ورودی‌ها و خروجی استفاده شده است.



مدار فوق با استفاده از نرم افزار *Proteus* رسم شده است. فایل مربوطه را در ضمیمه تقدیم می‌گردد.

توجه فرمایید که خروجی تابع مورد نظر به شکل زیر خواهد بود.

A	0	1	2	3	4	5	6	7	8	9
$f(A)$	2	52	102	152	602	652	702	752	1202	1252

توجه بفرمایید که ورودی و خروجی مدار به صورت BCD است. بنابرین میتوانیم ارقام خروجی را به تفکیک بررسی کنیم. همانگونه که مشخص است، برای نمایش خروجی نیاز به چهار رقم BCD داریم.

در تمامی حالات، یکان خروجی برابر ۲ است. بنابرین بدون نیاز به محاسبه همواره خروجی 0010 را در خروجی BCD یکان نمایش می‌دهیم.

خروجی دهگان به ازای Aهای زوج، 0000 و به ازای Aهای فرد 0101 است. بنابرین دهگان به عنوان تابعی از بیت اول ورودی BCD قابل بیان است. کافی است که بیت اول و سوم خروجی دهگان را به بیت اول ورودی وصل کنیم و بیت دوم و چهارم را به صفر وصل کنیم تا در صورتی که بیت اول ورودی ۱ باشد، خروجی برابر 0101 و در صورتی که برابر صفر باشد، خروجی برابر 0000 شود.

خروجی صدگان به ازای ورودی‌های مختلف به شکل زیر است.

A	A_3	A_2	A_1	A_0	O	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0
2	0	0	1	0	1	0	0	0	1
3	0	0	1	1	1	0	0	0	1
4	0	1	0	0	6	0	1	1	0
5	0	1	0	1	6	0	1	1	0
6	0	1	1	0	7	0	1	1	1
7	0	1	1	1	7	0	1	1	1
8	1	0	0	0	2	0	0	1	0
9	1	0	0	1	2	0	0	1	0

همان‌گونه که ملاحظه می‌فرمایید، خروجی به صورت دو عدد در میان یکسان است. بنابرین یکان ورودی A_0 در خروجی تاثیری ندارد و خروجی را میتوان به صورت تابعی از سه بیت $A_3A_2A_1$ نمایش داد.

همان‌گونه که ملاحظه می‌فرمایید، ستون O_0 دقیقا شکل ستون A_1 است و ستون O_2 دقیقا برابر ستون A_2 و ستون O_3 نیز همواره برابر صفر است بنابرین داریم

$$O_0(A_3, A_2, A_1) = A_1$$

$$O_2(A_3, A_2, A_1) = A_2$$

$$O_3(A_3, A_2, A_1) = 0$$

برای ساده‌سازی ستون O_1 از جدول کارنو استفاده می‌کنیم که به صورت زیر خواهد بود.

A_1 \ A_3A_2	00	01	11	10
0	**0**	**1**	***X***	**1**
1	**0**	**1**	***X***	***X***

$$O_1 = A_2 + A_3$$

خروجی هزارگان بر مبنای بیت چهارم ورودی قابل بیان است. زمانی که بیت پرارزش ورودی ۱ باشد، خروجی هزارگان ۱ و در غیر این صورت برابر صفر میشود. بنابرین کافی است که بیت اول خروجی هزارگان را به بیت چهارم ورودی وصل کنیم و بقیه بیت‌ها را به صفر وصل کنیم تا در صورتی که بیت پرارزش ورودی برابر ۱ باشد، خروجی هزارگان برابر 0001 و در غیر این صورت برابر 0000 شود.

سوال ۲

همانگونه که در سوال ۱ بخش مفاهیم اولیه بیان کردیم، قانون شنون یا بسط شنون (*Shannon Expansion Theory*) بیان میدارد که اگر f یک تابع منطقی از b_0، b_1 الی b_n باشد، خواهیم داشت

$$f(b_0, b_1, b_2, \dots, b_n) = b_0' f(0, b_1, b_2, \dots, b_n) + b_0 f(1, b_1, b_2, \dots, b_n)$$

توجه کنید که تابع $f(b_0, b_1, b_2, \dots, b_n)$ یک تابع با $n+1$ ورودی بود اما توابع $f(0, b_1, b_2, \dots, b_n)$ و $f(1, b_1, b_2, \dots, b_n)$ توابعی با n ورودی هستند و بدیهتا ساده‌سازی و پیاده‌سازی آنها ساده‌تر است.

می‌توانیم تابع مورد نظر را با استفاده از بسط شنون دو مرحله بسط دهیم. پس از این کار به رابطه زیر میرسیم.

$$f(w, x, y, z) = w' f(0, x, y, z) + w f(1, x, y, z)$$

$$= w'x' f(0,0, y, z) + w'x f(0,1, y, z) + wx' f(1,0, y, z) + wx f(1,1, y, z)$$

بدون از دست دادن کلیت، در نظر بگیرید که $wx = 00$ باشد. در این صورت خواهیم داشت

$$f(w, x, y, z) = w'x' f(0,0, y, z) + w'x f(0,1, y, z) + wx' f(1,0, y, z) + wx f(1,1, y, z)$$

$$\xrightarrow{wx=00} 1.f(0,0, y, z) + 0.f(0,1, y, z) + 0.f(1,0, y, z) + 0.f(1,1, y, z) = f(0,0, y, z)$$

به طور مشابه نتیجه میشود که

$$wx = 00 \Rightarrow f(w, x, y, z) = f(0, 0, y, z)$$

$$wx = 01 \Rightarrow f(w,x,y,z) = f(0,1,y,z)$$

$$wx = 10 \Rightarrow f(w,x,y,z) = f(1,0,y,z)$$

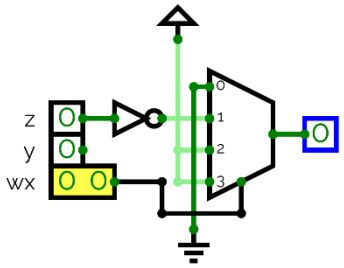
$$wx = 11 \Rightarrow f(w,x,y,z) = f(1,1,y,z)$$

بنابرین رابطه فوق را میتوانیم با استفاده از $MUX\ 4:1$ پیاده‌سازی کنیم. برای این منظور می‌توانیم wx را به خطوط انتخابی مولتی‌پلکسر متصل کنیم. سپس متناظر آن به خطوط ورودی توابع $f(0,0,y,z)$، $f(0,1,y,z)$، $f(1,0,y,z)$ و $f(1,1,y,z)$ را متصل کنیم. توجه کنید که چون این توابع دو ورودی دارند، ساده سازی آنها بسیار ساده‌تر است.

ساده‌سازی توابع به صورت زیر خواهد بود.

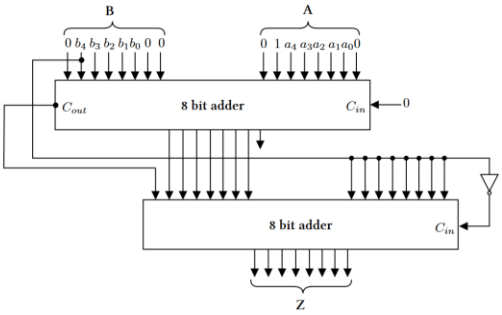
w	x	y	z	$f(w,x,y,z)$	$f^*(y,z)$
0	0	0	0	0	$= 0$
		0	1	0	
		1	0	X	
		1	1	0	
0	1	0	0	1	$= z'$
		0	1	0	
		1	0	X	
		1	1	0	
1	0	0	0	X	$= 1$
		0	1	1	
		1	0	1	
		1	1	1	
1	1	0	0	X	$= 1$
		0	1	1	
		1	0	1	
		1	1	1	

بنابرین مدار مورد نظر را می‌توانیم به شکل زیر بسازیم.



توجه کنید که ورودی wx یک ورودی ۲ بیتی است.

سوال ۳



توجه بفرمایید که خروجی مدار جمع‌کننده اول یک عدد ۸ بیتی به علاوه یک بیت $carry$ است. میتوانیم بیت $carry$ را به عنوان پرارزش‌ترین بیت در نظر بگیریم و خروجی این مدار جمع کننده را یک عدد ۹ بیتی در نظر بگیریم. یکی از ورودی های این جمع‌کننده عدد $4B$ و دیگری $2A + 32$ است. بنابرین خروجی این جمع‌کننده عدد $2A + 4B + 32$ می‌باشد. توجه کنید که چون کم‌ارزش‌ترین بیت هر دوی ورودی‌ها برابر صفر است، بیت کم‌ارزش خروجی نیز برابر صفر خواهد بود. خروجی این مدار با یک شیفت به راست، به ورودی جمع‌کننده بعدی متصل شده است. چون بیت کم‌ارزش خروجی برابر صفر است، شیفت به راست مانند تقسیم بر دو عمل می‌کند. بنابرین یکی از ورودی‌های جمع‌کننده دوم برابر $A + 2B + 16$ می‌باشد.

توجه کنید که در عمل چون بیت هشتم هر دوی ورودی‌های جمع‌کننده اول برابر صفر است، حاصل جمع ۸ بیتی نخواهد داشت و بیت $carry$ همواره برابر صفر خواهد بود.

اگر $B < 16$ باشد، b_3 برابر 0 خواهد بود. بنابرین ورودی دیگر جمع‌کننده دوم برابر صفر و ورودی c_{in} آن برابر ۱ خواهد بود. بنابرین خروجی جمع‌کننده دوم در این حالت برابر $A + 2B + 33$ خواهد بود. توجه کنید که چون c_{out} جمع‌کننده اول برابر صفر است، بیت هشتم ورودی دیگر برابر صفر خواهد بود و سرریز رخ نخواهد داد.

اگر $B \geq 16$ باشد، ، b_3 برابر 1 خواهد بود. بنابرین ورودی دیگر جمع‌کننده دوم برابر $11111111 = 2^8 - 1$ و ورودی c_{in} آن برابر 0 خواهد بود. بنابرین خروجی جمع‌کننده دوم در این حالت برابر $A + 2B + 32 + 2^8 - 1$ خواهد بود. از آنجایی که جمع را به صورت ۸ بیتی انجام می‌دهیم حاصل به پیمانه ۸ برابر $A + 2B + 32 - 1$ خواهد بود. بنابرین حاصل در این حالت برابر $A + 2B + 31$ خواهد بود.

بنابرین حاصل این مدار به صورت زیر خواهد بود.

IF (B < 16) THEN

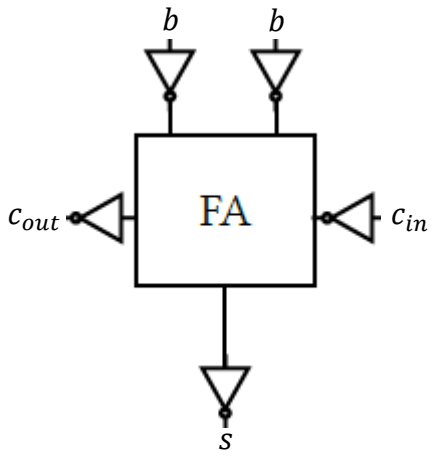
Z = A + 2B + 33

ELSE Z = A + 2B + 31

که معادل گزینه (b) است.

سوال ۴

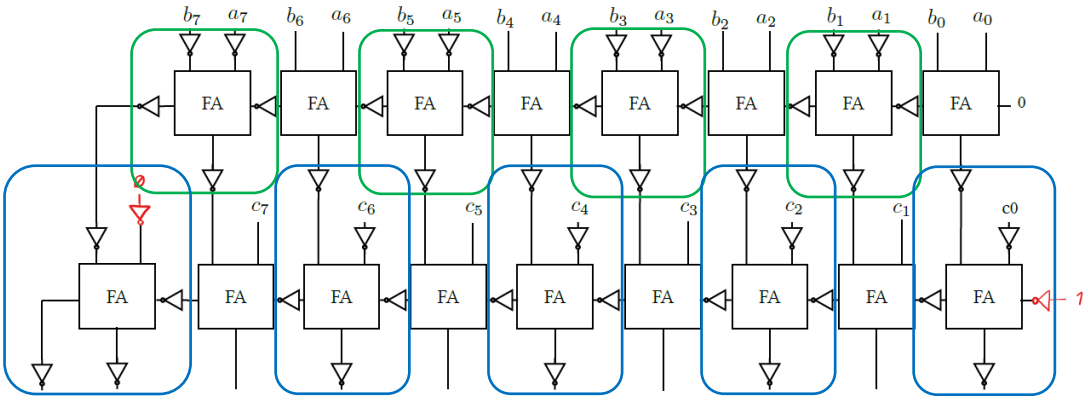
ابتدا میخواهیم خروجی مدار $Full\ Adder$ را در صورتی که مشابه شکل مقابل تمام ورودی ها و خروجی آن را Not کنیم بررسی کنیم. جدول حالات مدار مقابل به شکل زیر است.



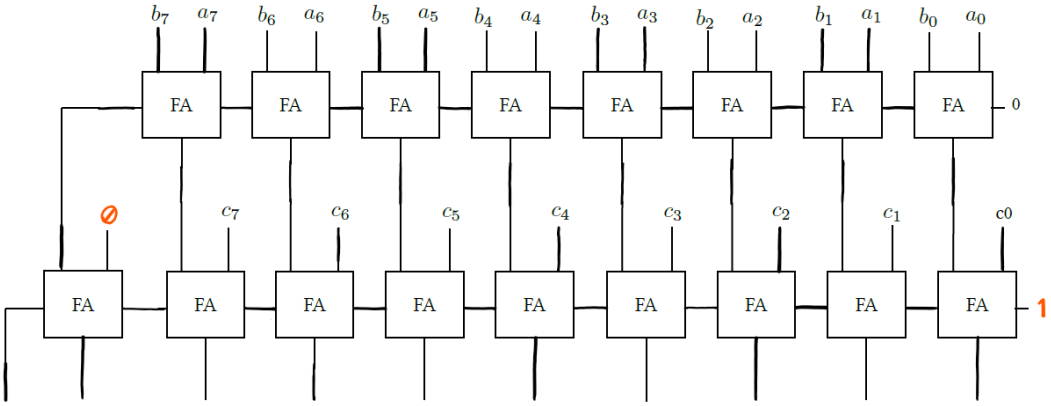
a	b	c_{in}	s	c_{cout}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

همانگونه که ملاحظه می‌کنید، جدول خروجی‌ها این مدار دقیقا مشابه $Full\ Adder$ عادی است. بنابرین تمام آنها را در مدار اصلی می‌توانیم با $Full\ Adder$ عادی جایگزین کنیم.

ابتدا توجه کنیدکه برای جایگزینی هرچه تمام تر، می‌توانیم ورودی c_{in} تمام جمع‌کننده پایین سمت راست و ورودی اول تمام‌جمع کننده پایین سمت چپ را نقیض و قبل از ورود آن به تمام جمع‌کننده یک گیت NOT اضافه کنیم. سپس تمام مدارات $Full\ Adder$ نقیض شده را با $Full\ Adder$ عادی جایگزین می‌کنیم. این جایگزین‌شونده‌ها را در شکل زیر مشخص کرده‌ایم.



پس از جایگزینی، مدار به شکل زیر در خواهد آمد.

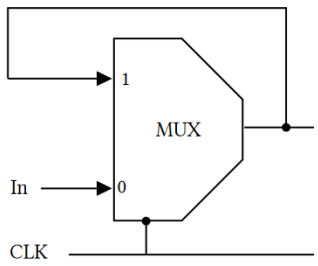


توجه کنید که در جمع ۸بیتی، خروجی ۸+۱ بیت است. یعنی به قراردادن $carry_{out}$ به عنوان پرارزش‌ترین بیت، خروجی ۹ بیتی حاصل میگردد.

در شکل بالا خروجی مرحله اول حاصل جمع ۹ بیتی A و B است. این عدد با ۹بیتی C که بیت پرارزش آن صفر است به همراه $carry$ ۱ جمع میشود و در خروجی ظاهر می‌گردد. بنابرین خروجی این مدار برابر $A + B + C + 1$ است. که معادل گزینه (آ) است.

مدارهای ترتیبی

سوال ۱



مدار مقابل را در نظر بگیرید. زمانی که ورودی کلاک برابر صفر است، ورودی *In* به خروجی متصل میگردد. زمانی که ورودی کلاک برابر ۱ باشد، خروجی مدار به خودش برمی‌گردد و دوباره در خروجی ظاهر می‌گردد. به عبارتی خروجی در مدار حفظ میگردد. این مدار عملکردی مشابه *Latch* دارد که حساسیت آن نسبت به کلاک به صورت *Level Triggered* می‌باشد.

حال می‌توانیم با استفاده از دو عدد *Latch* یک *FlipFlop* به کمک معماری *Master-Slave* بسازیم. برای این منظور کافیست که دو *Latch* را پشت سر یک دیگر قرار دهیم و خروجی اولی را به ورودی دومی وصل کنیم و کلاک را به اولی و نقیض کلاک را به دومی متصل کنیم. در زمانی که کلاک در وضعیت *low* قرار دارد، *Latch* اول در حالت بارگیری و *Latch* دوم در حالت ذخیره قرار میگیرد. بنابرین خروجی تغییر نمیکند ولی ورودی Latch دوم تغییر میکند. زمانی که کلاک از وضعیت *low* به وضعیت *high* می‌رود (*rising edge*)، *Latch* دوم در وضعیت بارگیری و *Latch* اول در وضعیت ذخیره قرار می‌گیرد. بنابرین آخرین ورودی ذخیره شده در Latch اول، که در آخرین زمانی که کلاک در وضعیت *low* قرار داشت یا به اصطلاح لحظه *rising edge* در آن ذخیره شده بود، در *Latch* دوم وارد و در خروجی ظاهر میشود. مادامی که کلاک در وضعیت *high* باشد، فلیپ‌فلاپ اول در وضعیت ذخیره است، بنابرین ورودی مدار عملا بی تاثیر است.

بنابرین مدار مورد نظر در زمان rising edge کلاک، ورودی را دریافت و در خود ذخیره می‌کند. در سایر زمان‌ها ورودی ذخیره شده را در خروجی نمایش می‌دهد.

سوال ۲

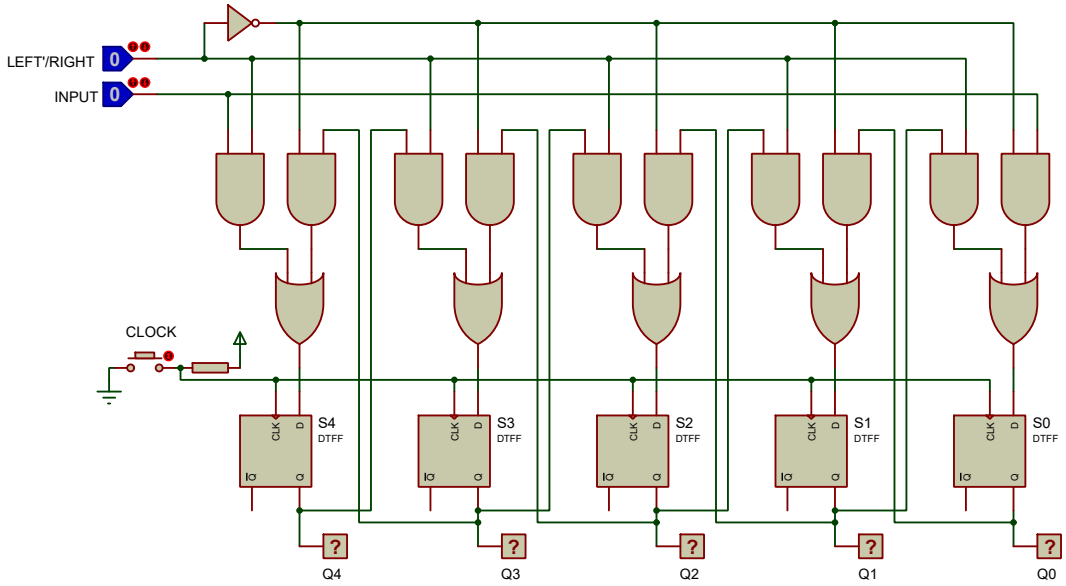
برای اینکه شیفت رجیستر قابلیت شیفت به راست و شیفت به چپ داشته باشد، باید یک ورودی کنترلی برای انتخاب داشته باشیم. به علاوه آن باید یک بیت ورودی برای بیتی که از سمت راست یا چپ وارد می‌شود نیز داشته باشم.

از آنجایی که قرار است این شیفت رجیستر به لبه کلاک حساس باشد، برای ساخت آن از فلیپ‌فلاپ که به لبه حساس است استفاده می‌کنیم.

توجه کنید که ورودی حالت بعدی هر یک از بیت‌ها (به استثنا بیت‌های مرزی) بسته به چپ یا راست بودن شیفت، بیت چپی یا بیت راستی است. در بیت‌های مرزی، به جای بیت کناری که وجود ندارد، ورودی را در نظر میگیریم. بنابرین کافی

است که از یک MUX برای ورودی هر یک از فلیپ‌فلاپ‌ها استفاده کنیم. به این صورت که ورودی کنترلی مدار را به ورودی انتخابی آن وصل کنیم و بیت چپ و راست را به تناظر به ورودی‌ها متصل کنیم. مقتضی بیت کنترلی بیت چپی یا راستی به ورودی فلیپ‌فلاپ‌ها متصل می‌گردد.

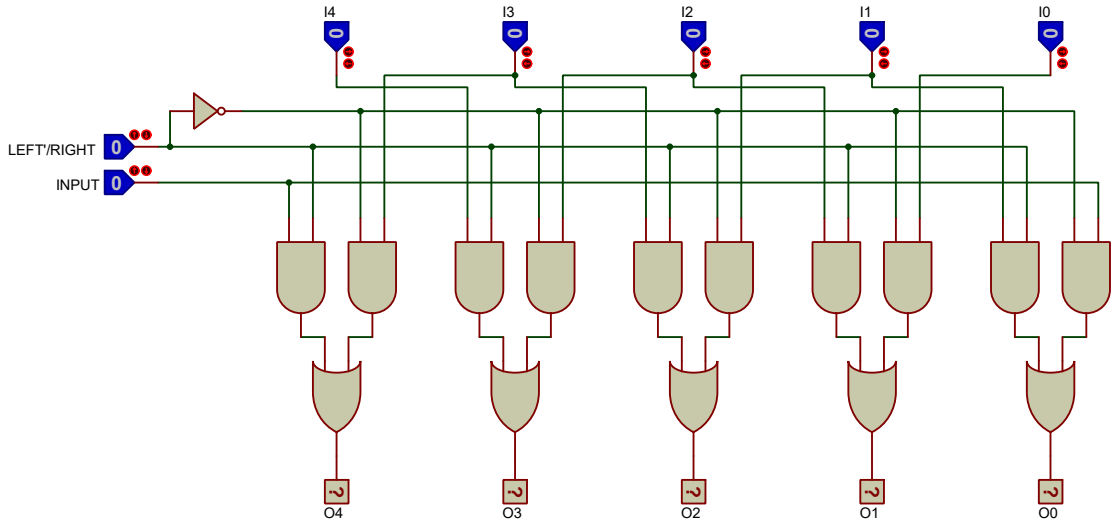
برای دقت بیشتر، MUX را در سطح گیت در شکل زیر پیاده‌سازی کرده‌ایم.



در مدار فوق، زمانی که ورودی کنترلی صفر باشد شیفت به سمت چپ و در صورتی که ۱ باشد شیفت به سمت راست خواهد بود. همچنین کلاک به صورت یک ***Push Button*** پیاده‌سازی شده است. در حالت عادی، مقدار خروجی از کلاک برابر یک است. با هر بار فشرده شدن و رها شدن دکمه، کلاک به زمین وصل و سپس قطع میگردد و مقدار کلاک به صفر و دوباره به یک برمیگردد. بنابرین یک لبه بالارونده و یک لبه پایین‌رونده بوجود میاید. از آنجایی که فلیپ‌فلاپ‌ها به صورت ***Rising edge triggered*** هستند، مقدار آنها تغییر خواهد کرد.

مدار فوق از دو بخش تشکیل شده است. چند فلیپ‌فلاپ که حالت مدار را ذخیره می‌کنند و یک مدار ترکیبی که با توجه به ورودی‌هایی که از حالت قبلی فلیپ‌فلاپ‌ها دریافت میکند و ورودی $input$ و کنترلی شیفت به راست و چپ، عدد دریافت شده را یک واحد به سمت راست و چپ شیفت می‌دهد. در حقیقت عمل $Shift$ خوردن توسط این مدار ترکیبی انجام می‌شود و فلیپ‌فلاپ‌ها تنها حافظ حالت مدار هستند.

توجه بفرمایید که $Shift\ Register$ و $Shifter$ یک تفاوت ماهوی با یکدیگر دارند. آن هم این است که $Shifter$ بر خلاف $Shift\ Register$ که یک مدار ترتیبی است، یک مدار ترکیبی است و حالتی را در خود ذخیره نمی‌کند. صرفا ورودی آن لحظه خود را با توجه به ورودی‌های جانبی شیفت و در خروجی نمایش می‌دهد. در شکل زیر میتوانید یک $Shift\ Register$ را مشاهده کنید.

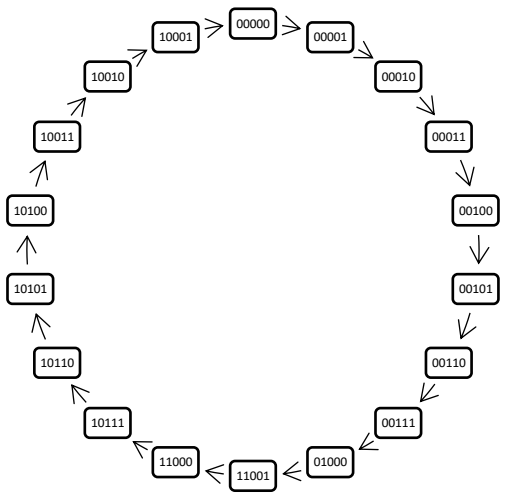


توجه فرمایید که مدار ترکیبی که به هر یک از بیت‌های خروجی وصل شده است، ساختار درونی یک $MUX\ 2:1$ است.

سوال ۳

در این مدار ترتیبی، ممکن است اعداد ۱ تا ۸ به صورت بالا یا پایین شمار و حالت صفر و ۹ ظاهر شوند. بنابرین ۱۸ حالت مختلف در این مدار داریم. بنابرین برای ساخت آن به حداقل $\lceil \log 18 \rceil = 5$ فلیپ‌فلاپ احتیاج داریم.

برای سادگی مراحل طراحی، معادل باینری ۵ بیتی $abcde$ را به این صورت تخصیص می‌دهیم که بیت پرارزش نشان دهنده وضعیت بالاشمار یا پایین‌شمار بودن مدار باشد به این شکل که صفر نشان دهنده بالاشمار بودن و ۱ نشان دهنده پایین شمار بود مدار باشد و ۴ بیت کم‌ارزش نشان دهنده عدد فعلی شمارنده باشند. معادل باینری حالات صفر و ۹ را نیز به صورت 00000 و 11001 در نظر میگیریم. بنابرین نمودار حالت به شکل زیر در خواهد آمد.



برای سادگی، از $D-FF$ استفاده می‌کنیم. همچنین طراحی را به صورت سنکرون انجام خواهیم داد. برای این منظور ورودی هر یک از فیلپ‌فلاپ‌ها را مورد بررسی قرار خواهیم داد. ساده‌سازی‌ها را مقتضی شرایط به صورت SOP و یا POS یا روش‌های دیگر انجام می‌دهیم تا حدالمقدور مدار ساخته شده ساده شود. در مواردی برای ساده‌سازی بیشتر از روش‌های دیگر استفاده می‌کنیم.

de \ bc	00	01	11	10
00	**0**	**0**	***X***	**1**
01	**0**	**0**	***X***	***X***
11	**0**	**0**	***X***	***X***
10	**0**	**0**	***X***	***X***

$a=0$

de \ bc	00	01	11	10
00	***X***	**1**	***X***	**1**
01	**0**	**1**	***X***	**1**
11	**1**	**1**	***X***	***X***
10	**1**	**1**	***X***	***X***

$a=1$

$$D_a = (a+b)(b+c+d)$$

de \ bc	00	01	11	10
00	0	0	X	1
01	0	0	X	X
11	0	1	X	X
10	0	0	X	X

$a=0$

de \ bc	00	01	11	10
00	X	0	X	0
01	0	0	X	1
11	0	0	X	X
10	0	0	X	X

$a=1$

$$D_b = a'b + be + a'cde$$

de \ bc	00	01	11	10
00	0	1	X	0
01	0	1	X	X
11	1	0	X	X
10	0	1	X	X

$a = 0$

de \ bc	00	01	11	10
00	X	0	X	1
01	0	1	X	0
11	0	1	X	X
10	0	1	X	X

$a = 1$

$$D_c = a'cd' + ace + a'c'de + cde' + abd'e'$$

de \ bc	00	01	11	10
00	0	0	X	0
01	1	1	X	X
11	0	0	X	X
10	1	1	X	X

$a = 0$

de \ bc	00	01	11	10
00	X	1	X	1
01	0	0	X	0
11	1	1	X	X
10	0	0	X	X

$a = 1$

$$D_d = a'd'e + a'de' + ad'e' + ade = a \oplus d \oplus e = XOR(a, d, e)$$

de \ bc	00	01	11	10
00	**1**	**1**	***X***	**1**
01	**0**	**0**	***X***	***X***
11	**0**	**0**	***X***	***X***
10	**1**	**1**	***X***	***X***

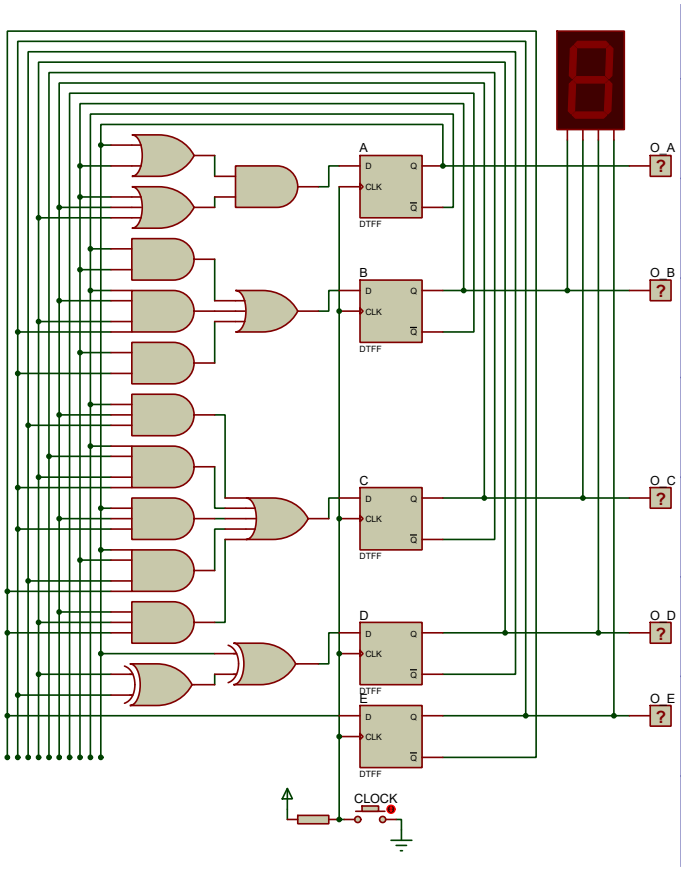
$a = 0$

de \ bc	00	01	11	10
00	***X***	**1**	***X***	**1**
01	**0**	**0**	***X***	**0**
11	**0**	**0**	***X***	***X***
10	**1**	**1**	***X***	***X***

$a = 1$

$$D_e = e'$$

بنابرین با توجه به روابط بالا، مدار را می‌توانیم به شکل زیر پیاده سازی کنیم.



توجه کنید که فلیپ‌فلاپ A برای نگهداری جهت شمارش و $BCDE$ برای نگهداری مقدار فعلی است. برای نمایش بهتر از یک نمایشگر $7-SEGMENT$ استفاده کرده‌ایم.

سوال ۴

ضرب دو عدد n بیتی X و Y را در نظر بگیرید. منظور از $\ll$، شیفت به چپ است.

$$X \times Y = x_n x_{n-1} x_{n-2} \dots x_0 \times y_n y_{n-1} y_{n-2} \dots y_0$$

$$= x_n x_{n-1} x_{n-2} \dots x_0 \times \sum_{i=0}^{n} (2^i \times y_i)$$

$$= \sum_{i=0}^{n} (2^i \times y_i \times x_n x_{n-1} x_{n-2} \dots x_0)$$

$$= \sum_{i=0}^{n} \left((y_i \times x_n x_{n-1} x_{n-2} \dots x_0) \ll i \right)$$

توجه کنید که اگر y_i برابر صفر باشد، حاصل $y_i \times x_n x_{n-1} x_{n-2} \dots x_0$ برابر صفر و اگر y_i برابر ۱ باشد، حاصل برابر $x_n x_{n-1} x_{n-2} \dots x_0$ خواهد بود. بنابرین برای محاسبه این مقدار به صورت منطقی، می‌توانیم که AND تمام بیت های X را با y_i را محاسبه کنیم. بنابرین حاصل در نهایت به صورت زیر خواهد بود. **(در AND زیر یکی از ورودی‌ها یک بیت y_i و دیگر تمام بیت‌های X است. در خروجی، AND تمام بیت‌های X و y_i، محاسبه می‌گردد.)**

$$X \times Y = \sum_{i=0}^{n} (AND(y_i, x_n x_{n-1} x_{n-2} \dots x_0) \ll i)$$

اساس کار مدار فوق نیز مشابه است. در هر مرحله، ضرب یک بیت از B را در کل A محاسبه می‌کند و با حاصلی که از قبل موجود است بعلاوه یک بیت از C جمع میکند و در هر مرحله حاصل یک واحد به سمت چپ شیفت میخورد. چون شیفت انجام میشود، حاصل نهایی عملیات بر روی A و B برابر ضرب این دو خواهد بود. همچنین چون یک بیت از C هر مرحله به عنوان بیت کم ارزش وارد میشود و شیفت میخورد، در نهایت عدد فوق با کل C نیز جمع می‌شود.

به بیان دقیق‌تر، در هر کلاک خروجی ذخیره شده در شیفت رجیستر به شکل زیر خواهد بود.

کلاک	مقدار ذخیره شده
0	0
1	$AND(B_3, A) + C_3 + 0$
2	$AND(B_2, A) + C_2 + \left((AND(B_3, A) + C_3) \ll 1\right)$
3	$AND(B_1, A) + C_1 + \left[\left(AND(B_2, A) + C_2 + \left((AND(B_3, A) + C_3) \ll 1\right)\right) \ll 1\right] =$ $AND(B_1, A) + C_1 + \left((AND(B_2, A) + C_2) \ll 1\right) + \left((AND(B_3, A) + C_3) \ll 2\right)$
4	$AND(B_0, A) + C_0 + \left[\left[AND(B_1, A) + C_1 + \left((AND(B_2, A) + C_2) \ll 1\right) + \left((AND(B_3, A) + C_3) \ll 2\right)\right] \ll 1\right] =$ $AND(B_0, A) + C_0 + \left((AND(B_1, A) + C_1) \ll 1\right) + \left((AND(B_2, A) + C_2) \ll 2\right) + \left((AND(B_3, A) + C_3) \ll 3\right)$

بنابرین در نهایت خروجی مدار به شکل زیر خواهد بود.

$$AND(B_0, A) + C_0 + \left((AND(B_1, A) + C_1) \ll 1\right) + \left((AND(B_2, A) + C_2) \ll 2\right) + \left((AND(B_3, A) + C_3) \ll 3\right) =$$

$$[(AND(B_0, A) \ll 0) + (AND(B_1, A) \ll 1) + (AND(B_2, A) \ll 2) + (AND(B_3, A) \ll 3)]$$
$$+ [(C_0 \ll 0) + (C_1 \ll 1) + (C_2 \ll 2) + (C_3 \ll 3)] =$$

$$\sum_{i=0}^{3}(AND(B_i, A_3A_2A_1A_0) \ll i) + C_3C_2C_1C_0 =$$

$$A_3A_2A_1A_0 \times B_3B_2B_1B_0 + C_3C_2C_1C_0$$

بنابرین خروجی معادل گزینه (آ) خواهد بود.