



دانشکده مهندسی برق

# مدارهای منطقی و سیستم‌های دیجیتال

آزمایش ۴ - طراحی، پیاده‌سازی و  
 تست مدارهای ترکیبی بر روی FPGA

# آزمایش ۴ - طراحی ، پیاده سازی و تست مدار های ترکیبی بر روی FPGA

در این آزمایش، ابتدا به راه اندازی 7-segment adder و نمایش خروجی بر روی 7-segment ها خواهیم پرداخت. در نهایت نیز با مفهوم hamming-code آشنا شده و یک encoder و decoder بر مبنای آن طراحی خواهیم کرد.

## قطعات و تجهیزات مورد نیاز

- بورد Zynq (AXPZ7010)

- Extension board

- یک عدد کابل USB و سیم پاور بورد

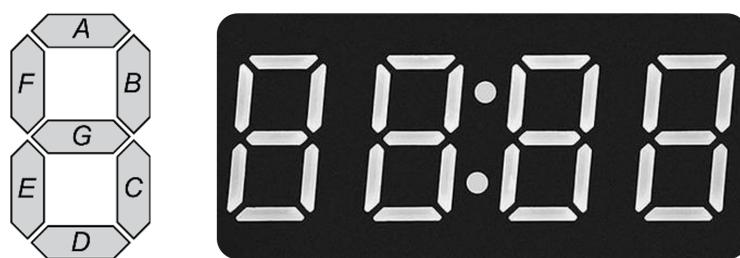
- نرم افزار Vivado 2019.1

## ۱- پیش‌مطالعه

در این بخش با نحوه استفاده از 7-seg 4 رقمی و extension board Hamming code و مفاهیم آشنا خواهید شد:

• 4-digit 7-seg :

در این قسمت به راه اندازی 7-seg چهار رقمی می‌پردازیم که از قرار زیر می‌باشد:



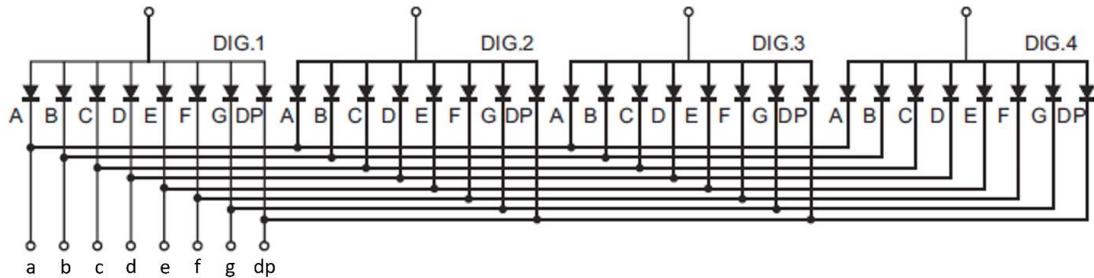
شکل ۱-۴: 7-seg چهار رقمی

این قطعه ۱۲ پایه دارد:

- ۷ پایه از آن به LED های a تا g هر چهار 7-seg متصل است.
- ۱ پایه به dp (decimal point) که در اینجا همان ":" است، متصل است.
- ۴ پایه به dig0, dig1, dig2, dig3 وصل اند که با یک شدن، آن رقم را روشن می‌کنند.

مدلی از 7-seg که ما در این آزمایش استفاده می‌کنیم، دارای مشخصات زیر است. دقت داشته باشید که مدل های مختلفی وجود دارند و باید برای آگاهی از این موارد، در صورت وجود دیتاباشیت، به آن مراجعه کنیم و در غیر این صورت از تست دیود مولتی متر کمک بگیریم:

- پایه های a تا g و dp "Active LOW" هستند.
- پایه های dig0 تا dig3 "Active HIGH" هستند.



شکل ۲-۴: مدار داخلی یک چهار رقمی 7-seg

با توجه به مدار فوق ، مشاهده می کنیم که اگر هر یک از Dig ها یک باشد و A-G و dp صفر ، جریان به سمت هر یک از LED های آن سگمنت ، شروع به حرکت می کند و آن را روشن می کند که نشان دهنده علت "Active HIGH" بودن Dig ها و "Active LOW" بودن پایه های G و dp است.

در تصاویر زیر ، دو حالت از نمایش اعداد را مشاهده می کنید. توصیه می شود که برای اینکه "بودن LED ها شما را دچار اشتباہ نکند ، ابتدا با فرض اینکه عدد مورد نظر "Active HIGH" است ، آن را نوشت و سپس قرینه کنید:



$$\{dig3, dig2, dig1, dig0\} = 0110$$

$$\{a, b, c, d, e, f, g\} = 0100100 = \sim(1011011)$$

$$dp = 1 = \sim(0)$$



$$\{dig3, dig2, dig1, dig0\} = 0001$$

$$\{a, b, c, d, e, f, g\} = 1001111 = \sim(0110000)$$

$$dp = 0 = \sim(1)$$

شکل ۳-۴: دو مثال از نمایش اعداد روی 7-seg چهار رقمی

### • Extension board •

برای پیاده سازی مداراتی که تعداد ورودی و خروجی زیادی دارند، نیازمند تعداد بالایی از کلید و LED هاست. بوردي که در دسترس شما قرار دارد، بوردي صنعتی است و از تعداد کلید و LED کمتری نسبت به بورد های آموزشی یا به اصطلاح Evaluation board بروخوردار است. برای حل این مسئله و امکان پیاده سازی مدارات پیشرفته تر، یک بورد کمکی برای شما فراهم شده است که به بورد قابلیت دسترسی به پریفرال های زیر را اضافه می کند:

10 LEDs —

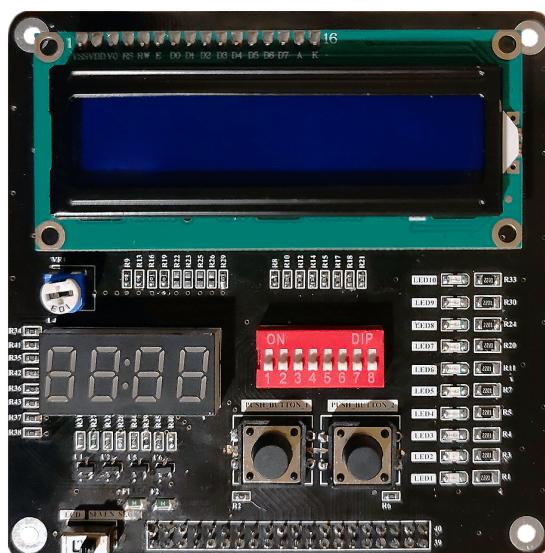
8 Dip switches —

2 Push buttons —

4-digit 7-seg —

LCD display —

بورد اصلی به صورت hardwired به برخی از پایه های تراشه زینک متصل شده است (مثلا 5 LED موجود بر روی بورد اصلی) و تعدادی از پین های آن، توسط شرکت سازنده آزاد قرار داده شده است تا کاربر بسته به نیاز خود از آن ها استفاده کند. دو مجموعه expansion port با مجموعا ۸۰ پین، در دو سمت بورد تعییه شده است که می توانند به عنوان خروجی یا ورودی به هر پریفرالی که مد نظر ما باشد متصل شود.



شکل ۴-۴: نمایی از بورد کمکی

در این جدول، اتصال پین های بورد کمکی به پین های تراشه زینک را مشاهده می کنید:

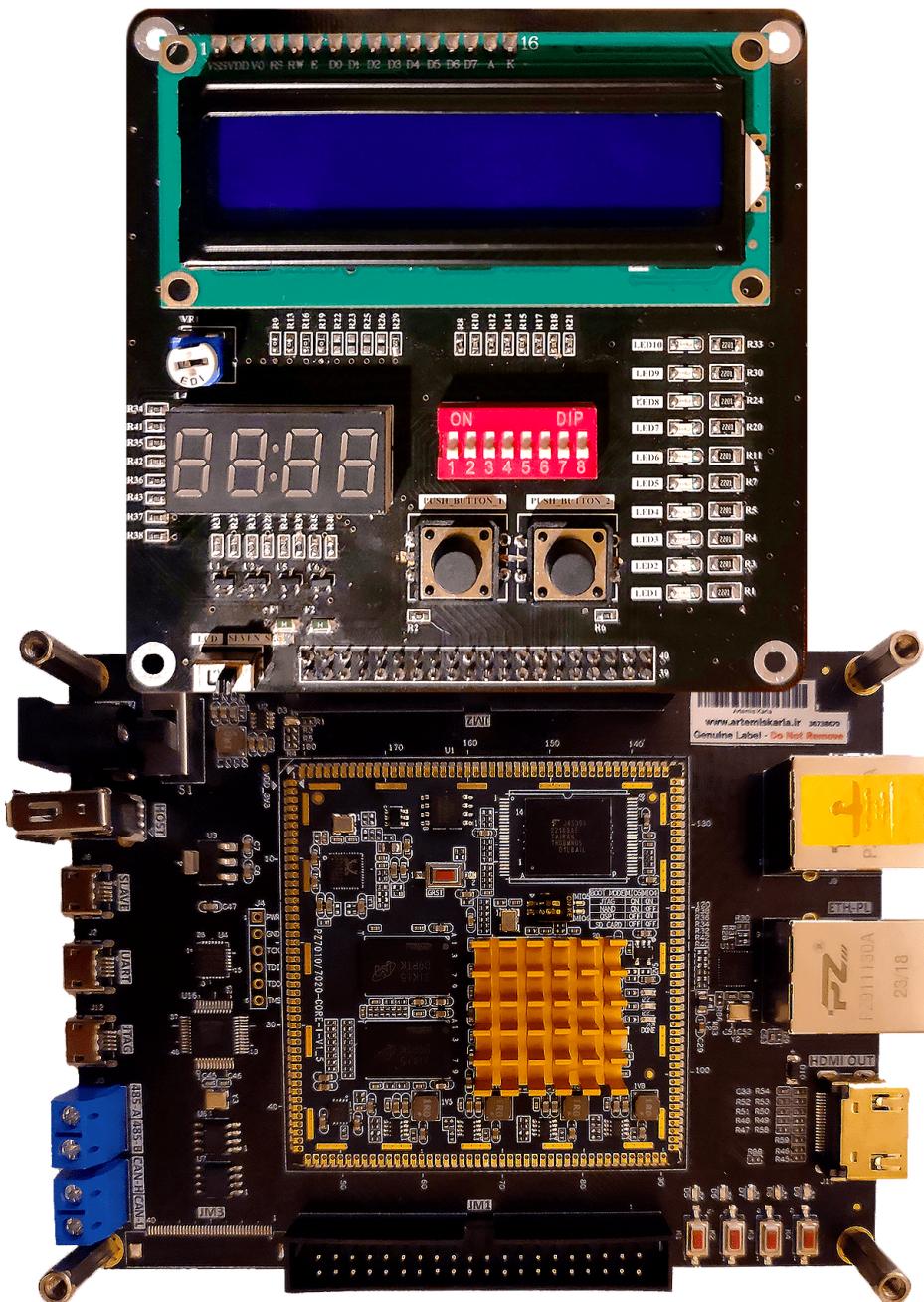
Pinout					
Id.	Ext. board	Zynq	Id.	Ext. board	Zynq
1	Vdd 5V	Vdd 5V	21	Vdd 3.3V	Vdd 3.3V
2	GND	GND	22	GND	GND
3	7-seg "dig3" LCD "RS"	B19	23	7-seg "dig2" LCD "RW"	G20
4	7-seg "dig1" LCD "EN"	D19	24	7-seg "dig0"	D20
5	7-seg "a" LCD "D0"	F16	25	7-seg "b" LCD "D1"	F17
6	7-seg "c" LCD "D2"	H15	26	7-seg "d" LCD "D3"	G15
7	7-seg "e" LCD "D4"	J18	27	7-seg "f" LCD "D5"	H18
8	7-seg "g" LCD "D6"	H16	28	7-seg "dp" LCD "D7"	H17
9	Push button1	J20	29	Push button2	H20
10	DIP 1	K14	30	DIP 2	J14
11	DIP 3	M14	31	DIP 4	M15
12	DIP 5	L19	32	DIP 6	L20
13	DIP 7	M19	33	DIP 8	M20
14	LED 10	N15	34	LED 9	N16
15	LED 8	C20	35	LED 7	B20
16	LED 6	E17	36	LED 5	D18
17	GND	GND	37	GND	GND
18	GND	GND	38	GND	GND
19	LED 4	E18	39	LED 3	E19
20	LED 2	G17	40	LED 1	G18

شکل ۴-۵: اتصالات پین های بورد کمکی به پین های تراشه زینک

نکات مهم :

- LED های ۱ تا ۱۰، dip switch ها و push button ها همگی "Active HIGH" هستند.
- "Active LOW" هستند اما g-a و dp همگی "Active HIGH" Dig3-Dig0 هستند.
- سوییچی که در گوش پایین و سمت چپ بورد قرار دارد، همواره در مود 7-seg باشد.
- بورد کمکی برای اتصال صحیح می بایست تنها و تنها از طریق پین های JM2 روی بورد اصلی نصب شود.

بورد کمکی متصل شده بر بورد اصلی در نهایت به شکل زیر در می آید:



شکل ۴-۶: بورد کمکی نصب شده بر روی بورد اصلی

### : Hamming code

در بخشی از آزمایش باید مدار Error Correcting Code (ECC) را به روش [7, 4, 3] Hamming در محل A ارسال کنید. فرض کنید قصد داریم چهار بیت را از محل A به محل B ارسال کنیم. با توجه به اینکه ممکن است در مسیر ارسال یکی از بیت‌ها تغییر کند (نویز یا هر عامل مزاحم)، سه بیت اضافی به همراه دیتای اصلی به B ارسال می‌شود. این سه بیت حاوی اطلاعاتی است که در صورت تغییر یکی از داده‌ها، B می‌تواند آن را تشخیص و سپس محل آن را شناسایی کرده و در نتیجه دیتا را بازیابی نماید. به این سه بیت، بیت‌های parity (در فارسی: بیت برابری یا توازن) گفته می‌شود. برای پیاده‌سازی این روش نیازمند دو مدار هستیم. یک مدار فرستنده در محل A که بیت‌های parity را تولید می‌کند، و یک مدار گیرنده در محل B که با توجه به هفت بیت دریافتی، بیت معیوب را شناسایی و کد ارسالی را بازسازی می‌کند.

برای اطلاعات بیشتر در مورد Hamming code و به صورت کلی و [7, 4, 3] Hamming که مورد بحث این آزمایش است به ترتیب به [این ویدیو](#) و [این سایت](#) مراجعه کنید.

## ۲- پیش‌گزارش

- پیش مطالعه را بخوانید و با نحوه کارکرد بورد کمکی ، 7-seg hamming code آشنا شوید.
- متن دستور کار را مطالعه کرده و کد وریلاگ مورد نیاز تمامی بخش ها را بنویسید.  
(دقت داشته باشد که در شروع آزمایش، موجود بودن کد بخش های مختلف توسط دستیاران بررسی می شود و تنها دانشجویانی که کد ها را زده باشند اجازه انجام آزمایش را خواهند داشت).

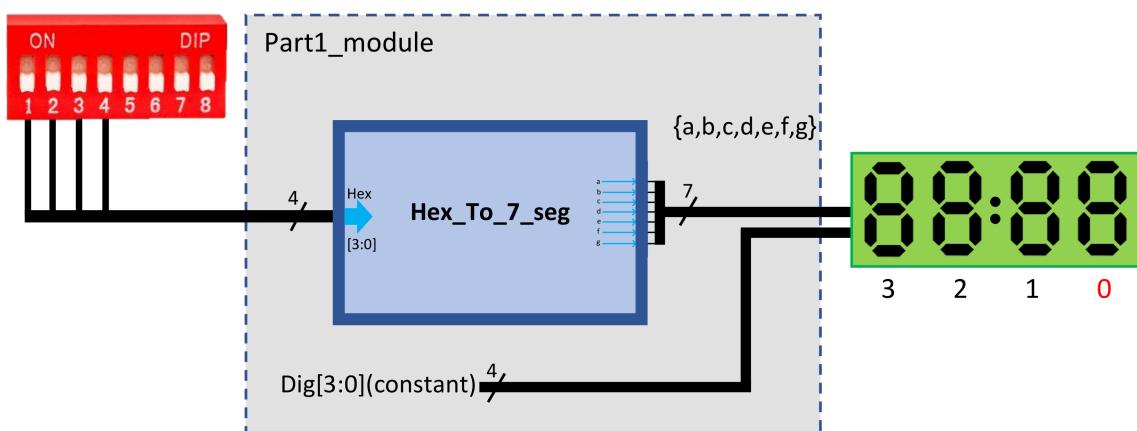
### ۳- دستور کار

#### ۱-۳ بخش اول (زمان تقریبی: ۱۵ دقیقه)

در این بخش به نحوه استفاده از 7-seg مسلط می شویم تا در بخش های بعدی از آن استفاده کنیم. کد ماژولی را بنویسید که در ورودی عددی ۴ بیتی (از dip switch ها) را گرفته و آن را در قالب عدد بر روی dig0 نمایش دهد. یعنی عدد ۱۰ تا ۱۵ را به صورت A تا F hex نمایش دهد.

برای سهولت کار، یک ماژول Hex\_to\_7\_seg به شما داده شده است که در ورودی آن یک عدد ۴ بیتی گرفته می شود و در خروجی، پتن مورد نیاز برای LED های a تا g سون سگمنت جهت نمایش عدد مورد نظر قرار می گیرد.

در نهایت مدار شما می بایست به شکل زیر باشد :



شکل ۷-۴: شماتیک مدار بخش اول



در این بخش و دیگر بخش ها، هیچ تغییری در ماژول Hex\_to\_7\_seg اعمال نکنید!

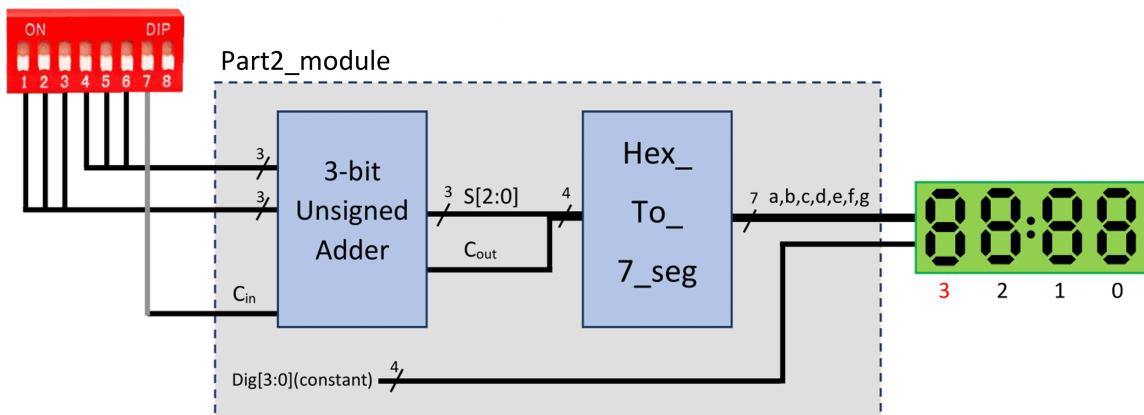
```
module Hex_to_7_seg
(
    input wire [3:0] Hex,
    output reg a,b,c,d,e,f,g
);

    always @ (Hex) begin
        case(Hex)
            0 : {a,b,c,d,e,f,g} = ~(7'b1111110);
            1 : {a,b,c,d,e,f,g} = ~(7'b0110000);
            2 : {a,b,c,d,e,f,g} = ~(7'b1101101);
            3 : {a,b,c,d,e,f,g} = ~(7'b1111001);
            4 : {a,b,c,d,e,f,g} = ~(7'b0110011);
            5 : {a,b,c,d,e,f,g} = ~(7'b1011011);
            6 : {a,b,c,d,e,f,g} = ~(7'b1011111);
            7 : {a,b,c,d,e,f,g} = ~(7'b1110000);
            8 : {a,b,c,d,e,f,g} = ~(7'b1111111);
            9 : {a,b,c,d,e,f,g} = ~(7'b1111011);
            10 : {a,b,c,d,e,f,g} = ~(7'b1110111); // A
            11 : {a,b,c,d,e,f,g} = ~(7'b0011111); // B
            12 : {a,b,c,d,e,f,g} = ~(7'b1001110); // C
            13 : {a,b,c,d,e,f,g} = ~(7'b0111101); // D
            14 : {a,b,c,d,e,f,g} = ~(7'b1001111); // E
            15 : {a,b,c,d,e,f,g} = ~(7'b1000111); // F
        endcase
    end

endmodule
```

### ۲-۳ بخش دوم (زمان تقریبی: ۳۰ دقیقه)

در آزمایش ۳ ماژول full adder و half adder instance گرفتن از این دو ماژول یک مدار بایستی unsigned adder را پیاده سازی کردید. حال با Dip3-Dip1 ورودی یک سه بیتی طراحی کنید. مدار بایستی یک ورودی را از Dip6-Dip4، ورودی دیگر را از Dip7 و carry in Dip6-Dip4 بگیرد. حاصل جمع چهار بیتی (سه بیت خروجی جمع کننده ها به همراه بیت carry out) را بر Dig3 در قالب Hex نمایش دهد.

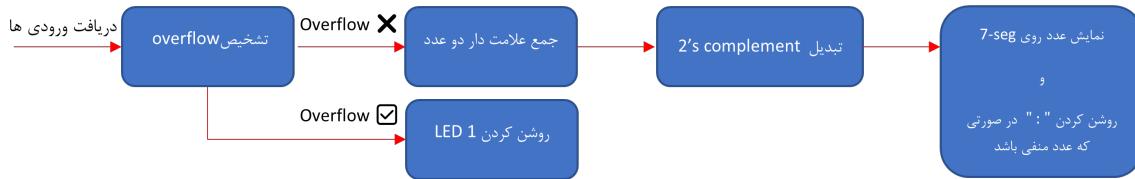


شکل ۴-۸: شماتیک مدار بخش دوم

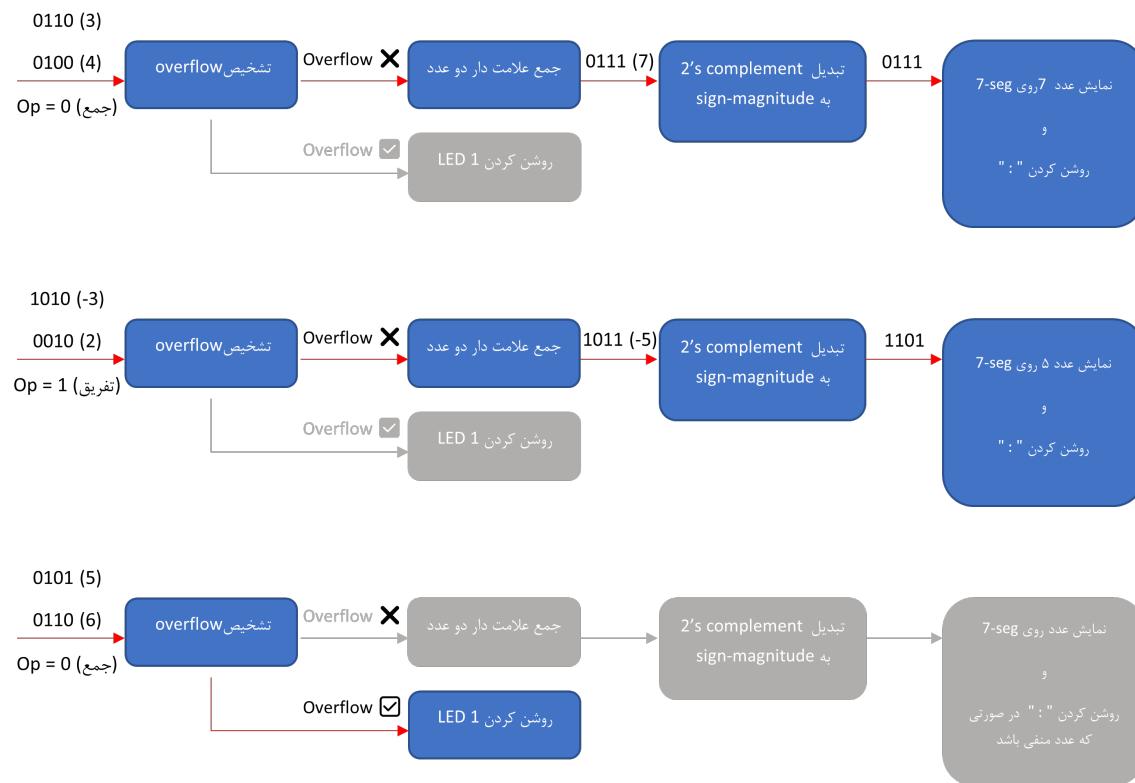
### ۳-۳ بخش سوم (زمان تقریبی: ۴۵ دقیقه)

حال با بکارگیری چهار full adder و مدارهای مورد نیاز دیگر، یک ماژول جمع و تفریق کننده ۴ چهار بیتی علامت‌دار طراحی کنید. (carry in را به صورت hardwired به صفر وصل کنید تا صرفاً دو عدد ۴ بیتی علامت دار در ورودی قرار گیرد). یک بیت op برای برای مدار خود در نظر بگیرید که با صفر بودن آن، عملیات جمع و با یک بودن آن، عملیات تفریق انجام شود. همچنین مدار تشخیص سرریز (overflow) را به آن اضافه کنید. چنانچه حاصل سرریز داشت، از آنجا که نمایش ۴ بیت جواب حاصل به صورت Hex بی معنا است، تنها LED1 بورد کمکی را به نشانه وقوع این پدیده روشن کنید. در صورتی که سرریز رخ نداد، حاصل چهار بیتی علامت‌دار خروجی را پس از تبدیل به فرمت sign-magnitude بر روی Dig0 از سون سگمنت ها نشان دهد. برای نمایش علامت منفی، از "—" یا همان dp استفاده کنید.

برای فهم بهتر از روند دریافت ورودی/خروجی و چند مثال، شکل ۹-۴ و ۱۰-۴ را ملاحظه کنید.



شکل ۴-۹: روند دریافت ورودی ها و نمایش خروجی



شکل ۴-۱۰: چند مثال ورودی و خروجی دریافتی مدار بخش سوم

### ورودی ها و خروجی ها:

- دو ورودی چهار بیتی را به کمک Dip8-Dip1 و Dip4-Dip5 اعمال کنید.
- بیت op را به کمک push button1 مقداردهی کنید.
- خروجی بر روی 7-ha نمایش داده شود و در صورت منفی بودن dp را روشن نماید.
- در صورت رخداد overflow ، LED1 از بورد کمکی را روشن کنید.

## مقدمه بخش چهارم تا ششم :

با خوانش بخش پیش مطالعه، با مفهوم Hamming Code آشنا شدید. در بخش چهارم تا ششم می خواهیم خواهیم با ساز و کار مدار زیر آشنا شویم. در ابتدا به روش hamming ۳ بیت پریتی برای داده ۴ بیتی ورودی محاسبه و به آن اضافه می شود و عددی ۷ بیتی از مازول encoder خارج می شود. سپس با تشکیل مازول transmission line ، یکی از این ۷ بیت را فرینه می کنیم تا تاثیر نویز بر سیگنال را شبیه سازی کنیم. در نهایت داده ۷ بیتی نویزی را به مازول decoder می دهیم تا بیت خراب را پیدا و اصلاح کند. سپس ۴ بیت داده را استخراج و در خروجی خود تحویل دهد :



شکل ۱۱-۴: مدار یک سیستم انکودر و دیکودر hamming

### ۴-۳ بخش چهارم (Encoder) (زمان تقریبی: ۳۰ دقیقه)



شکل ۴: مدار انکودر

برای تولید بیت‌های parity و اضافه کردن آنها به ورودی چهار بیتی  $x_0x_1x_2x_3$  باید به شکل زیر عمل نمود:

$$\text{Encode: } x_0x_1x_2x_3 \rightarrow p_0p_1x_0p_2x_1x_2x_3, \quad \text{where} \quad p_0 = x_0 \oplus x_1 \oplus x_3, \\ p_1 = x_0 \oplus x_2 \oplus x_3, \\ p_2 = x_1 \oplus x_2 \oplus x_3.$$

مدار انکودر را طراحی و بر روی بورد پیاده‌سازی کنید. برای ورودی از چهار کلید Dip4-Dip1 استفاده کنید و کد هفت بیتی حاصل را بر روی LED7-LED1 بورد کمکی نمایش دهید. سپس به ازای همهٔ ترکیب‌های ۱۶ گانه ورودی، هفت بیت خروجی را با جدول زیر مقایسه کنید. (بیت های قرمز رنگ پریتی بیت‌ها و بیت‌های سیاه رنگ بیت‌های دیتا هستند)

Input	Output	Input	Output	Input	Output	Input	Output
0000	0000000	0100	1001100	1000	1110000	1100	0111100
0001	1101001	0101	0100101	1001	0011001	1101	1010101
0010	0101010	0110	1100110	1010	1011010	1110	0010110
0011	1000011	0111	0001111	1011	0110011	1111	1111111

شکل ۴: جدول ورودی و خروجی‌های انکودر

### ۵-۳ بخش پنجم (Decoder) (زمان تقریبی: ۳۰ دقیقه)



شکل ۱۴-۴ مدار انکوڈر hamming

در بخش قبل encoder ای طراحی کردید که که قادر به اضافه کردن بیت های پریتی به ۴ بیت داده ی ورودی بود. حال در این بخش فرض می کنیم که دیتای نویز دار به دست ما رسیده است و ما می خواهیم با ساخت یک decoder بیت خراب را شناسایی کنیم. سپس بیت خراب را اصلاح کرده و در خروجی ۴ بیت دیتای اصلی را استخراج کرده و همچنین در ۳ بیت، شماره بیت خراب را نمایش دهیم.

شما به صورت دستی ۷ بیت داده دریافت شده را بر روی Dip7-Dip1 تنظیم می کنید و سپس مازول شما می بایست این ۷ بیت را به صورت ورودی دریافت کند و بررسی کند که با توجه به بیت های پریتی سنت شده در ۷ بیت دریافتی آیا خطای در بیتی رخ داده یا خیر. چنانچه خطای وجود داشت، آن بیت را اصلاح کند و ۴ بیت دیتا را بر روی LED4-LED1 بورد کمکی نمایش دهد.

چنانچه داده ورودی خطای نداشت ، عدد ۰ را بر روی Dig0 سون سگمنت ها نمایش دهید و چنانچه بیت شماره n ام دچار خطأ شده بود، عدد n را نمایش دهید.

**Decode:** Say we receive  $p_0'p_1'x_0'p_2'x_1'x_2'x_3'$ .  
 $(p_i' = p_i, x_j' = x_j \text{ if no errors})$

$$\begin{aligned} \text{Let } c_0 &= p_0' \oplus x_0' \oplus x_1' \oplus x_3', \\ c_1 &= p_1' \oplus x_0' \oplus x_2' \oplus x_3', \\ c_2 &= p_2' \oplus x_1' \oplus x_2' \oplus x_3'. \end{aligned}$$

$(c_2c_1c_0)_{10}$	Decode to
3	$\overline{x_0'}x_1'x_2'x_3'$
5	$x_0'\overline{x_1'}x_2'x_3'$
6	$x_0'x_1'\overline{x_2'}x_3'$
7	$x_0'x_1'x_2'\overline{x_3'}$
other values	$x_0'x_1'x_2'x_3'$



ترکیب  $c_2c_1c_0$  محل بیت احتمالاً خراب را نشان می‌دهد. اگر هیچ خطایی رخ نداده باشد، بر اساس روابط بالا، این مقدار صفر خواهد بود، اما در صورت خرابی حداکثر یکی از بیت‌ها، محل آن توسط این عدد نمایش داده می‌شود. از آنجایی که برای ما تنها بیت‌های چهارگانه‌ی اصلی اهمیت دارند، باید طبق جدول مقابل، چهار بیت اصلی ارسالی از طرف فرستنده را استخراج کرده و بیت خراب را complement کرد. در مورد بقیه‌ی مقادیر عدد  $c_2c_1c_0$  یعنی (۱، ۲، ۴)، خرابی در بیت parity رخ داده است. این بیت‌ها اگرچه در تشخیص خطا به ما کمک کرده‌اند، ولی به عنوان بخشی از پیام ارسالی نبوده و لذا در جدول به ازای این مقادیر، تغییری در کد دریافتی اعمال نمی‌شود. مجدداً یادآوری می‌شود صفر بودن این مقدار به معنای عدم وقوع هرگونه خطایی است.

$$\begin{array}{l} \begin{array}{l} 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \\ P_0'P_1'X_0'P_2'X_1'X_2'X_3' \end{array} \xrightarrow{\left\{ \begin{array}{l} C_0 = P_0' \wedge X_0' \wedge X_1' \wedge X_3' = 1 \wedge 0 \wedge 1 \wedge 1 = 1 \\ C_1 = P_1' \wedge X_0' \wedge X_2' \wedge X_3' = 0 \wedge 0 \wedge 0 \wedge 1 = 1 \\ C_2 = P_2' \wedge X_1' \wedge X_2' \wedge X_3' = 1 \wedge 1 \wedge 0 \wedge 1 = 1 \end{array} \right\}} (C_2C_1C_0)_{10} = 7 \xrightarrow{\text{بیت 7 ام یا همان } X_3 \text{ دچار خطأ شده است.}} \\ \xrightarrow{\text{استخراج بیت‌های دیتا}} 0100 \xrightarrow{\text{نمایش دیتا بر روی LED ها}} \square \blacksquare \blacksquare \blacksquare \end{array}$$

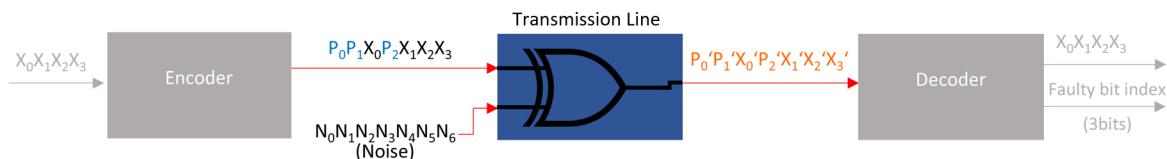
$$\begin{array}{l} \begin{array}{l} 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \\ P_0'P_1'X_0'P_2'X_1'X_2'X_3' \end{array} \xrightarrow{\left\{ \begin{array}{l} C_0 = P_0' \wedge X_0' \wedge X_1' \wedge X_3' = 0 \wedge 0 \wedge 0 \wedge 0 = 0 \\ C_1 = P_1' \wedge X_0' \wedge X_2' \wedge X_3' = 0 \wedge 0 \wedge 0 \wedge 1 = 0 \\ C_2 = P_2' \wedge X_1' \wedge X_2' \wedge X_3' = 1 \wedge 1 \wedge 0 \wedge 1 = 0 \end{array} \right\}} (C_2C_1C_0)_{10} = 0 \xrightarrow{\text{هیچ یک از بیت‌ها دچار خطأ نشده است.}} \\ \xrightarrow{\text{استخراج بیت‌های دیتا}} 0010 \xrightarrow{\text{نمایش دیتا بر روی LED ها}} \square \blacksquare \blacksquare \blacksquare \xrightarrow{\text{نمایش شماره صفر روی 7-seg}} 0 \end{array}$$

شکل ۴-۱۵ چند مثال از ورودی و خروجی‌های دریافتی دیکودر

برای انجام تست‌های بیشتر از **جدول بخش ۴** کمک بگیرید. برای تست ورودی‌های خراب، یک بیت از عدد ۷ بیتی جدول را قرینه کنید و به عنوان ورودی به مازول خود بدهید و ببینید آیا ۴ بیت دیتا و شماره بیت خراب به درستی پیدا می‌شود یا خیر.

! تاکید می‌شود که در این روش، تنها می‌توان یک بیت معیوب را شناسایی و اصلاح کرد، بنابراین از خراب کردن هم‌زمان بیش از ۱ بیت خودداری کنید!

### ۳-۶ بخش ششم (Transmission Line) (زمان تقریبی: ۳۰ دقیقه)



شکل ۱۶-۴ مدار

در این بخش به طراحی transmission line می پردازیم که قرار است نقش رابط بین encoder و decoder را ایفا کند. این مژول در ورودی دو عدد ۷ بیتی دریافت می کند و آن ها را به صورت bitwise با یکدیگر xor می کند. تا زمانی که  $N_0$  تا  $N_6$  همگی صفر باشند، حاصل xor آن با ورودی دیگر، همان ورودی دیگر خواهد بود. به محض اینکه یکی از بیت های  $N_0$  تا  $N_6$  یک شوند، آن بیت از ورودی دیگر به صورت قرینه در خروجی قرار خواهد گرفت. در حقیقت گیت xor یک inverter قابل برنامه ریزی است به این گونه که با یک کردن بیت  $n$  ام در ورودی دوم، خروجی برابر با ورودی اولی خواهد بود که بیت  $n$  ام قرینه شده است. به این شکل یک خطای ساختگی بر روی داده های خروجی encoder را شبیه سازی می کنیم. حال برای تست مژول transmission line و دیگر مژول ها در کنار هم، آن ها را به شکل زیر به هم متصل کنید:



شکل ۱۷-۴ مدار کامل متشکل از ۳ مژول بخش های قبل

یک عدد چهار بیتی توسط Dip4-Dip1 encoder به مژول Dip5-Dip7 وارد می شود. جهت شبیه سازی وقوع خطأ در انتقال اطلاعات، هفت بیت خروجی encoder به عنوان ورودی اول transmission line، با هفت بیت  $\{Dip5, Dip6, Dip7, Key0, Key1, Key2, Key3\}$  به عنوان نویز و ورودی دوم، xor می شود. بدین ترتیب با فشردن هر یک از این کلیدها، بیت مربوطه بترتیب از صفر و یک به یک و صفر تغییر وضعیت داده و گویی در انتقال آن خطایی رخ داده است. اکنون این هفت بیت پس از xor شدن وارد decoder می شوند و چهار بیت اصلی کد را بر روی LED4-LED1 بورد کمکی و شماره بیت معیوب را بر روی Dig0 سون سگمنت نمایش داده می شود.



بدین ترتیب، عدد معادل چهار کلید ورودی باید عیناً به LED4-LED1 بورد کمکی منتقل گردد. در آغاز تست می بایست عدد صفر بر روی Dig0 سون سگمنت نمایان شود، چرا که خطایی رخ نداده است. چنانچه یکی از کلید های ست شده جهت ایجاد خطا فشرده شوند، دیتای نمایش داده شده بر روی LED4-LED1 می بایست ثابت بماند و تنها عدد نمایان بر روی Dig0 سون سگمنت به شماره بیت معیوب شده تغییر کند.

(منظور از Key ها کلید های فشاری موجود بر روی بورد اصلی است، دقت داشته باشد که این کلید ها "Active LOW" هستند و چنانچه قرینه نشوند، در حالتی که هیچ دکمه ای فشرده نشده است، ۴ خط ایجاد می کنند که اصلاً مطلوب نیست)

### ۷-۳ بخش هفتم (امتیازی)

می دانیم که روش Hamming code قادر به تشخیص و اصلاح تنها یک بیت معیوب در دیتای ارسال شده است. اما این روش همچنان می تواند وجود دو بیت معیوب را شناسایی کند ولی قادر به پیدا کردن مکان آن ها در دیتای ارسالی نیست فلذًا امکان اصلاح آن را ندارد. بررسی کنید که از چه طریق می توان به وجود دو بیت معیوب پی برد و با تغییر ۳ ماظول قبلی ، سناریوی زیر را پیاده سازی کنید:

- در صورتی که بیت معیوبی وجود نداشت ، ۴ بیت دیتا همچنان بر روی LED4-LED1 بورد کمکی و عدد صفر بر روی Dig0 سون گمنت ها نمایش داده شود.
- در صورت وجود یک بیت معیوب ، ۴ بیت دیتا بدون تغییر نسبت به بیت های ورودی بر روی LED4-LED1 و نمایش داده شود و شماره بیت معیوب بر روی Dig0 سون سگمنت نمایش داده شود.
- در صورت وجود دو بیت معیوب ، با توجه به اینکه امکان اصلاح وجود ندارد، حرف "E" یا همان عدد ۱۴ Hex بر روی Dig0 و Dig1 به معنای وجود دو ارور نمایان شده و LED4-LED1 همگی روشن شوند.

در صورت نیاز به کلید های بیشتر ، در استفاده از باقی دکمه های بورد اصلی/کمکی آزاد هستید.

### ۳- گزارش

- مزیت استفاده از پریتی ها چیست که حاضر به ارسال ۳ بیت اضافه بر ۴ بیت داده هستیم؟
- چطور به روش hamming code قابل به تشخیص وجود دو خطأ خواهیم بود؟ چرا در این حالت قادر به پیدا کردن مکان بیت های معیوب نمی باشیم؟
- در خصوص آنچه از این آزمایش آموختید گزارش ۱ الی ۲ صفحه‌ای (مطابق **تمپیت** ارجاع شده در ابتدای این داکیومنت) بنویسید.  
دیدگاه خود نسبت به بخش‌های مختلف آزمایش را به انضمام پیشنهادهایتان بنویسید.