به نام خدا

پروژه مدارهای منطقی

(نیمسال دوم ۱۳۹۹-۱۴۰۰)

مقدمه

- هدف از این پروژه، پیاده‌سازی برخی قطعات مدارهای منطقی به زبان توصیف VHDL می‌باشد که در نهایت یک واحد محاسبانی ALU پیاده خواهد شد.
- نمره پروژه، مازاد بر نمره اصلی می‌باشد.
- انجام بخشی از پروژه نیز دارای نمره به همان میزان می‌باشد.
- هر دانشجو باید یک گزارش کار به همراه فایل‌های اجرایی پروژه خود را انجام دهد.

پروژه شامل ۳ بخش می‌شود؛ بخش اول پیاده سازی تمام جمع‌کننده، بخش دوم پیاده‌سازی گذرگاه داده و بخش سوم واحد محاسبات منطقی.

سه بخش موردنظر که شما باید انجام داده و ارسال نمایید با تیترهای **گزارش بخش اول پروژه**، **گزارش بخش دوم پروژه** و **گزارش بخش سوم پروژه**، مشخص شده‌اند.

- درصورت مشاهده کپی در پروژه‌ها، نمره کل پروژه تقسیم بر تعداد تکرار (کپی)های یک پروژه می‌شود و به تمامی دانشجوهای صاحب آن پروژه، نمره تقسیم شده داده می‌شود.

آشنایی با نرم افزارISE با مثال

شما در این دوره چگونگی طراحی توسط نرم افزار Xilinx ISE را بطور دقیق , با جزئیات کامل می‌آموزید و یاد می‌گیرید چگونه با انواع روشهای طراحی با تراشه‌های شرکت Xilinx کار کنید . مراحل زیر را با آرامش خاطر دنبال کنید . . .

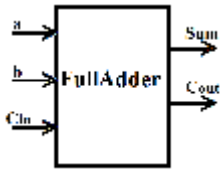
۱-۶. ساختار یک برنامه VHDL

برای توصیف هر قطعه سخت افزاری باید یک فایل VHDL ایجاد کرد. این قطعه سخت افزاری می‌تواند یک گیت خیلی ساده باشد تا یک پردازنده. هر فایل VHDL از سه بخش تشکیل شده است: معرفی کتابخانه ها، تعریف اینترفیس قطعه با دنیای بیرون خود، توصیف عملکرد قطعه.

برای معرفی کتابخانه از دستورات Library و use استفاده می شود. به عنوان نمونه، سه خط زیر در اغلب فایلهای VHDL آورده می شود.

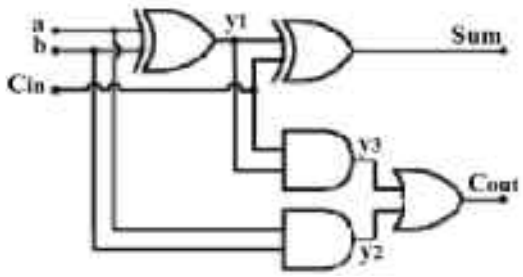
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_UNSIGNED.ALL;
```

برای تعریف اینترفیس قطعه با دنیای بیرون خود از کلمه کلیدی entity استفاده می شود. به عنوان مثال برای یک تمام جمع کننده می توان این اینترفیس را به صورت زیر تعریف کرد. (معمولا نامی که برای فایل انتخاب می شود با نامی که برای entity انتخاب می شود یکسان است)




```
entity FullAdder is
        Port (
                a :in std_logic;
                b :in std_logic;
                Cin :in std_logic;
                Sum :out std_logic;
                Cout :out std_logic
                        );
end FullAdder;
```

برای بیان توصیف عملکرد قطعه، از کلمه architecture استفاده می‌شود. در تعریف یک architecture باید مشخص شود که این توصیف برای کدام entity نوشته می‌شود.



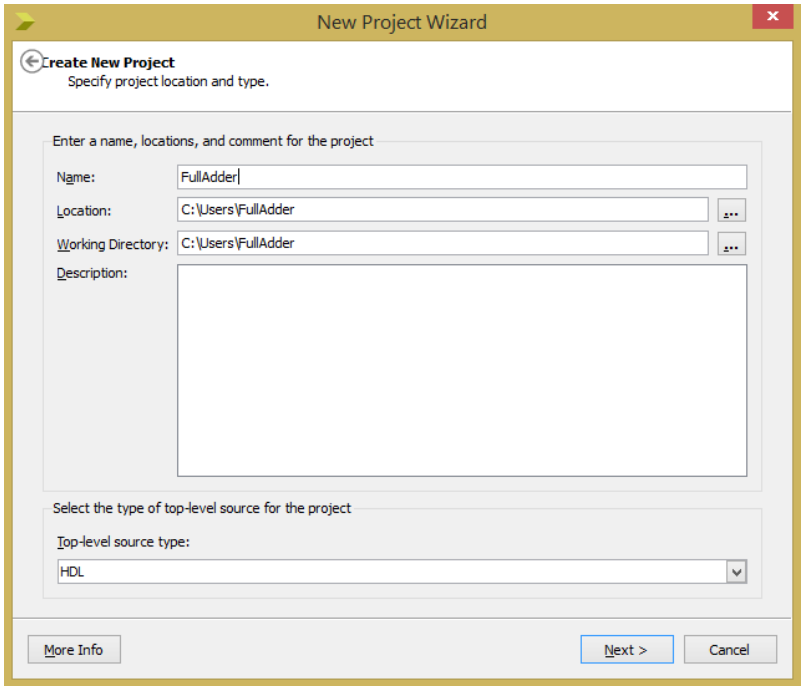
به عنوان مثال توصیف عملکرد داخلی یک تمام جمع کننده به صورت زیر خواهد بود:

```
architecture Behavioral of FullAdder is
  signal y1,y2,y3:std_logic:='0';
begin
        y1 <= a xor b;
        Sum <= y1 xor Cin;
        y2<= a and b;
        y3<= y1 and Cin;
        Cout<= y2 or y3;
end Behavioral;
```

۲-۶. آشنایی با نرم افزار ISE

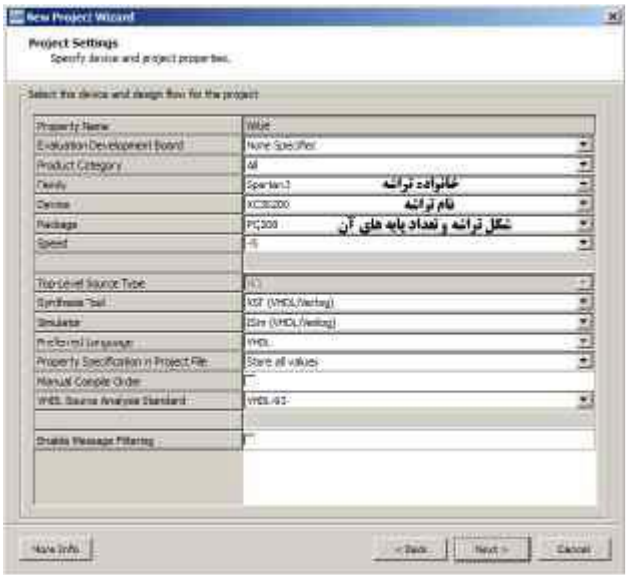
نرم‌افزار ISE از محصولات شرکت زایلینکس است که یک محیط مجتمع برای کار با FPGA می باشد. این نرم افزار قابلیت شبیه سازی، سنتز و ایمپلیمنت و پراگرام کردن تمام تراشه های این شرکت را دارد. در این آزمایش مراحل کامل نوشتن یک برنامه VHDL و شبیه سازی آن برای یک تمام جمع کننده توسط این نرم افزار آموزش داده می شود.

از منوی file گزینه New Project را انتخاب کنید. پنجره شکل ۶-۱ نمایش داده می شود. در این پنجره در قسمت Name نام پروژه را وارد کنید. (مثلا FullAdder).



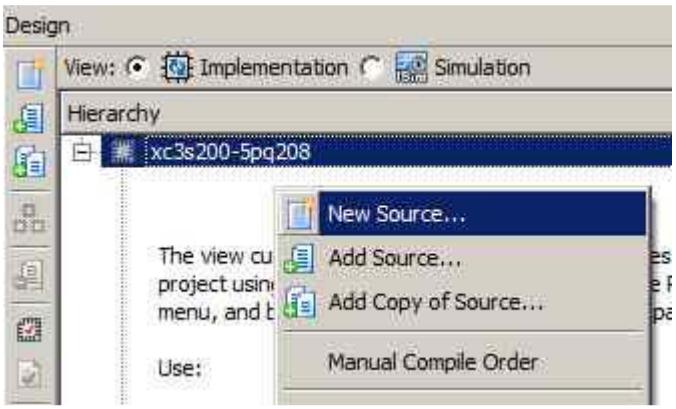
شکل ٦-١: ایجاد یک پروژه جدید

در مرحله بعد باید نام و مشخصات FPGA را معین کرد. در شکل ۶-۲ مشخصات یکی از FPGAهایی که در آزمایشگاه استفاده می شود، انتخاب شده است.



شکل ٦- ٢: انتخاب FPGA

بعد از اتمام ایجاد پروژه، باید فایلهای VHDL خود را به پروژه اضافه کنید. برای ایجاد یک برنامه VHDL جدید، مطابق شکل ۶-۳ روی نام تراشه FPGA کلیک راست نموده و گزینه New Source را انتخاب کنید.



شکل ٦- ٣: ایجاد یک فایل جدید و اضافه کردن آن به پروژه

سپس در پنجره ظاهر شده، نوع فایل جدید را VHDL انتخاب کنید و نام فایل را وارد کنید. در مرحله بعد نام پورتها و نوع آن را مشخص کنید (با توجه به توضیحات داده شده در بخش قبلی) تا نرم افزار ISE فایل VHDL را ایجاد کند.

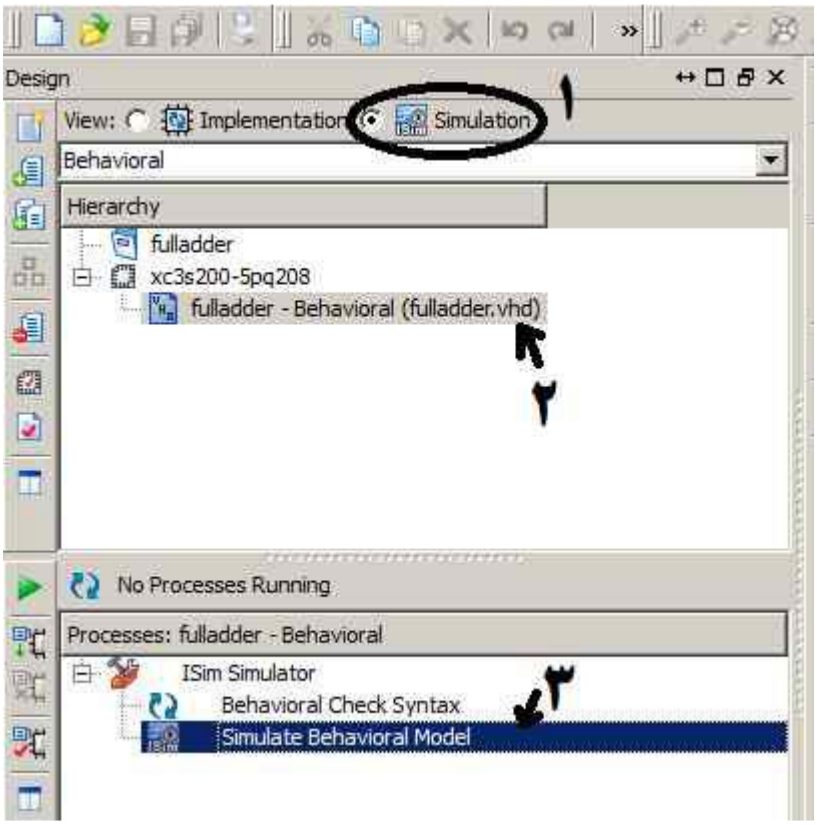
شکل ٦- ٤: انتخاب نوع فایل جدید و نامگذاري آن

بعد از اتمام مراحل فوق، فایل VHDL تولید شده نمایش داده می شود. در این فایل کتابخانه های مورد نیاز به فایل اضافه شده و همچنین Entity نیز به صورت اتوماتیک تولید شده است. تنها کاری که

باید انجام شود این است که قسمت architecture نیز تکمیل گردد. طبق برنامه داده شده در بخش قبل، این قسمت را نیز تکمیل کنید.

۳-۶. شبیه‌سازی مدار

برای وارد شدن به محیط شبیه سازی، باید سه گام انجام دهید. در شکل ۶-۵ این سه گام نمایش داده شده است. در گام اول گزینه Simulation را انتخاب کنید.



شکل ٦- ٤: انتخاب گزینه Simulation

در گام دوم، روی نام فایل مورد نظر کلیک کنید (ممکن است در پروژه چندین فایل VHDL وجود داشته باشد که در این صورت با کلیک کردن روی نام فایل مورد نظر تعیین می کنید که قصد شبیه سازی کدام فایل را دارید).

در گام سوم، روی گزینه Simulate Behavioral Model، دابل کلیک کنید تا پنجره ISIM باز شود. در این گام به پنجره کنسول و پیغامهای نمایش داده شده نیز دقت کنید ممکن است در برنامه شما اشکالاتی وجود داشته باشد که پیغامهای آن در پنجره کنسول نمایش داده می‌شود.

در برنامه ISIM شما می توانید با کلیک راست روی نام سیگنالها و پورتها و انتخاب گزینه Force Constant... به سیگنالهای خود مقدار صفر یا یک را اختصاص دهید و شبیه سازی کنید.

۴-۶. نوشتن برنامه VHDL برای قطعات پیچیده

توصیف سیستمهای سخت افزاری پیچیده توسط VHDL، عموما به صورت سلسله مراتبی انجام می شود. روش سلسله مراتبی به این صورت است که طرح بزرگ به طرحهای کوچکتر شکسته می شود و کد VHDL طرحهای کوچکتر نوشته می شود سپس با ترکیب آنها طرح بزرگتر ایجاد می گردد. به عنوان مثال برای برای پیاده سازی یک جمع کننده ۲ بیتی می توان ابتدا کد جمع کننده یک بیتی (تمام جمع کننده یا Full adder) را نوشت سپس با کنار هم قرار دادن ۲ تا از آنها می توان جمع کننده ۲بیتی را ایجاد کرد. در ادامه برنامه مورد نیاز برای ساخت یک جمع کننده ۲ بیتی با استفاده از ۲عدد FullAdder آورده شده است.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Adder2Bits is
	Port (
		a :in std_logic_vector(1 downto 0);
		b :in std_logic_vector(1 downto 0);
		Cin :in std_logic;
		Sum :out std_logic_vector(1 downto 0);
		Cout :out std_logic
			);
end Adder2Bits;
architecture Behavioral of Adder2Bits is
component FullAdder is
	Port (
		a :in std_logic;
		b :in std_logic;
		Cin :in std_logic;
		Sum :out std_logic;
		Cout :out std_logic
			);
end component;
signal c1:std_logic:='0';
begin
U0: FullAdder
	Port map (
```

```
                a => a(0),
                b => b(0),
                Cin => Cin,
                Sum => Sum(0),
                Cout => c1
        );
U1: FullAdder
    Port map (
                a => a(1),
                b => b(1),
                Cin => c1,
                Sum => Sum(1),
                Cout => Cout
        );
end Behavioral;
```

۵-۶. گزارش بخش اول پروژه

برنامه VHDL برای یک جمع کننده ۴ بیتی بنویسید به نحوی که از ۴ عدد fulladder استفاده شده باشد.

یک فولدر به نام Adder4bits ایجاد کنید در داخل آن کل پروژه "آی اس ای" مربوط به آن را قرار دهید سپس آن را زیپ کنید.

نکته: برای اینکه حجم فایلهای پروژه کاهش یابد، قبل از ارسال، گزینه CleanUp project files را از منوی project انتخاب کنید تا فایلهای موقتی را حذف کند.

بخش دوم؛ پیاده‌سازی باس مشترک

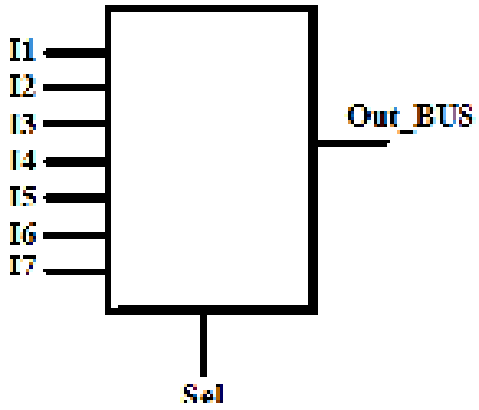
در این بخش نحوه پیاده سازی مالتی پلکسر و استفاده از آن برای پیاده سازی باس مشترک و همچنین نحوه پیاده سازی مدارهای ترتیبی دنبال می‌شود.

۱-۷. روش پیاده سازی باس مشترک

در شکل ۷-۱، باس مشترک استفاده شده در کامپیوتر پایه نمایش داده شده است. یکی از روشهای ایجاد باس مشترک، استفاده از مالتی پلکسر می‌باشد. در زبان VHDL، برای پیاده سازی مالتی پلکسر می توان از دستور انتساب شرطی استفاده کرد. در قطعه برنامه زیر نحوه پیاده سازی یک مالتی پلکسر ۴ در ۱ با زبان VHDL نمایش داده شده است.

```
Outp <= I0 when Sel=”00”else
        I1 when Sel=”01”else
        I2 when Sel=”10”else
        I3 ;
```

ساختار باس مشترکی که در کامپیوتر پایه استفاده می شود، به صورت زیر است. در آزمایشگاه ماجولی به نام Common_BUS ایجاد کنید و برنامه آن را بنویسید.



شکل ۷-۱: ورودي و خروجی هاي یک باس مشترك

۷-۲. روش پیاده سازی مدارهای ترتیبی در VHDL

مدارهای ترتیبی در زبان VHDL را با مفهومی به نام Process پیاده سازی می کنند. ساختار کلی Process به صورت زیر است:

```
Process(Clk)
Begin
        If (Clk=’1’ and Clk’event) then

        End if;
End process;
```

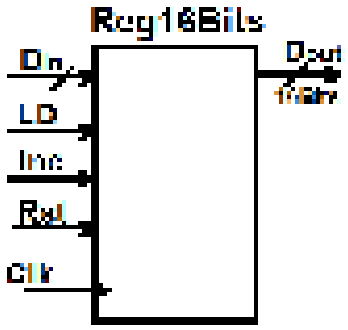
دستور if که در process فوق به کار برده شده است، بیانگر این است که دستورات داخل بدنه if زمانی اجرا شود که لبه بالارونده کلاک رخ دهد. اگر مدار به لبه پایین رونده کلاک حساس باشد آنگاه دستور Clk='1' باید به دستور Clk='0' تبدیل شود.

دستورات داخل Process به صورت ترتیبی اجرا می شوند. در ادامه نحوه پیاده سازی دو مدار ترتیبی که برای پیاده سازی کامپیوتر پایه مورد نیاز است، آورده می شود.

۷-۳. پیاده‌سازی یک رجیستر ۱۶ بیتی

رجیسترهایی که در کامپیوتر پایه استفاده می شوند دارای مشخصات زیر هستند: حساس به لبه بالارونده، دارای قابلیت load شدن، دارای قابلیت Increament و دارای قابلیت Clear شدن.

Entity مربوط به رجیستر ۱۶ بیت را می توان به صورت شکل ۷-۲ در نظر گرفت.



شکل ۷-۲: ورودي و خروجی هاي یک رجیستر ۱۶ بیتی

کد مربوط به رجیستر ۱۶ بیتی به صورت زیر می‌باشد.

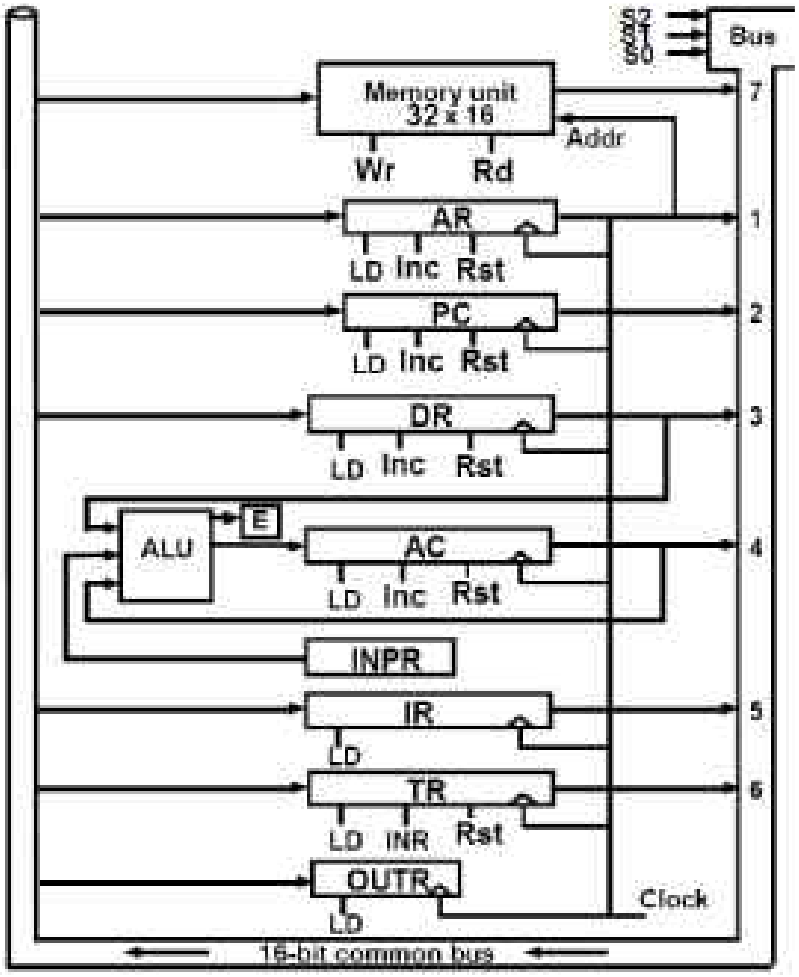
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_UNSIGNED.ALL;

entity Reg16Bits is
	Port (
		Clk : in STD_LOGIC;
		Rst : in STD_LOGIC;
		LD : in STD_LOGIC;
		Inc : in STD_LOGIC;

		Din : in STD_LOGIC_VECTOR(15 downto 0);
		Dout : out STD_LOGIC_VECTOR(15 downto 0)
			);
end Reg16Bits;

architecture Behavioral of Reg16Bits is
signal Dout_sig: STD_LOGIC_VECTOR(15 downto 0);
begin
	Dout <= Dout_sig;

	process(Clk)
	begin
		if(Clk='1' and Clk'event) then
			if(Rst='1')then
				Dout_sig<=(others=>'0');
			elsif(LD='1')then
				Dout_sig<= Din;
			elsif(Inc='1')then
				Dout_sig<= Dout_sig+1;
			end if;
		end if;
	end process;
end Behavioral;
```

نکته ۱: در برنامه فوق، اولویت با Rst، سپس با LD ونهایتا با Inc در نظر گرفته شده است. یعنی اگر همزمان این سه پایه با همدیگر فعال شدند اولویت به Rst داده می‌شود.

نکته ۲: با توجه به اینکه سیگنال Dout خروجی است نمیتوان برای افزایش آن از دستور Dout <= Dout+1; استفاده کرد، به همین دلیل یک سیگنال ۱۶ بیتی به نام Dout_sig تعریف شده است که عملیات روی آن انجام می شود و نهایتا با استفاده از دستور Dout <= Dout_sig مقدار آن به خروجی انتساب داده می شود.

۴-۷. گزارش بخش دوم پروژه

در این پروژه قصد داریم با استفاده از برنامه های رجیستر و باس مشترک که در جلسه گذشته داشتیم، برنامه VHDL مربوط به شکل۷-۳ را بنویسیم.



شکل ۷-۳: ساختار باس مشترك و DataPath در کامپیوتر پایه کتاب موریس مانو

۱- یک برنامه به نام Mano_DataPath بنویسید که دیاگرام شکل۷-۳ را با استفاده از کامپوننتهای Common_Bus و Reg16 پیاده‌سازی کند (قسمتهای ALU و Memory و فلیپ فلاپ E در جلسه بعد توضیح داده خواهد شد).

راهنمایی: سیگنالهایی که در شکل۷-۳ نمایش داده شده است به دو دسته تقسیم می شوند: دسته اول، سیگنالهایی هستند که به صورت داخلی می باشند و دسته دوم، سیگنالهایی هستند که باید به صورت پورتهای ورودی یا خروجی تعریف شوند.

پورتهای ورودی ماژول Mano_DataPath عبارتند از:

الف) تمام پایه های کنترلی رجیسترها (این پایه های کنترلی توسط واحد کنترل تولید خواهند شد و به ماژول Mano_DataPath خواهند آمد)

ب) خطوط انتخاب باس مشترک

ج) خطوط خواندن و نوشتن حافظه RAM

د) ۸ بیت ورودی برای به نام INPR

ه) هفت بیت به نام Control_Cmd که از واحد کنترل برای ALU می‌آید.

و) سیگنال کلاک و سیگنال Reset

ز) پایه های ریست ، مکمل ساز و لود برای فلیپ فلاپ E

پورتهای خروجی ماژول Mano_DataPath عبارتند از: ۸ بیت برای رجیستر OUTR، یک بیت برای E، ۱۶ بیت خروجی رجیستر IR (برای ارسال دستورالعمل به واحد کنترل)، ۱۶ بیت خروجی رجیستر AC، ۱۶ بیت خروجی رجیستر DR.

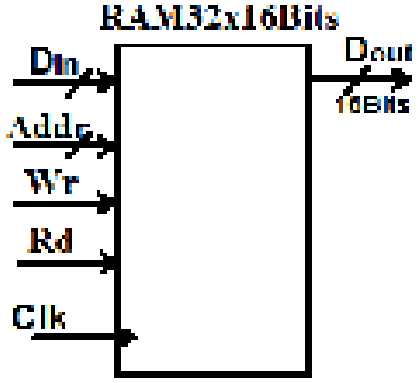
پروژه ISE را که شامل برنامه‌های نوشته شده در جلسه قبل و فایل Mano_DataPath است با نام بخش دوم پروژه ذخیره و ارسال کنید.

بخش سوم؛ ادامه پیاده سازی DataPath

بلوک DataPath در کامپیوتر پایه دارای ۶ زیر بخش بود. در بخش قبل با زیر بخشهای Reg16 ، Common_Bus آشنا شدید در این بخش با نحوه پیاده سازی زیربخش Memory، فلیپ فلاپ E و ALU آشنا خواهید شد. همچنین با مفهوم TestBench و نحوه استفاده از آن برای شبیه سازی سخت افزار آشنا خواهید شد.

۸-۱. پیاده سازی یک حافظه RAM

حافظه استفاده شده در کامپیوتر پایه به صورت ۴۰۹۶ کلمه ۱۶ بیتی است. برای پیاده سازی حافظه در زبان VHDL می توان از آرایه استفاده کرد. از آنجایی که آرایه بعد از پیاده سازی حجم وسیعی از منابع FPGA را اشغال می کند به جای پیاده سازی یک حافظه ۴۰۹۶ کلمه ای یک حافظه ۳۲ کلمه-ای پیاده سازی می‌کنیم. در شکل ۸-۱ ورودی ها و خروجی های حافظه RAM نمایش داده شده است.



شکل ۸-۱: ورودي و خروجی هاي حافظه RAM

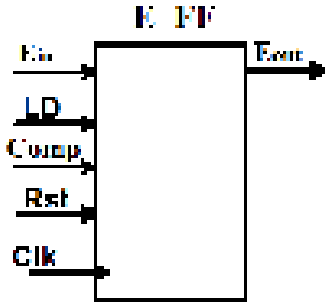
نکته پیاده سازی: در FPGAهای امروزی، بلوکهای مخصوص حافظه وجود دارد که به Block RAM معروف هستند و در صورت استفاده از آنها، منابع داخلی FPGAها هدر نمی رود. (استفاده از آرایه برای پیاده سازی حافظه باعث هدر رفتن منابع FPGAها و کند شدن فرایند سنتز و ایمپلیمنت خواهد شد).

برنامه VHDL برای توصیف یک حافظه ۳۲ کلمه ای که هر کلمه آن ۱۶ بیتی است به صورت زیر است.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_unsigned.all;

entity RAM32x16Bits is
	Port (
		Addr	: in	STD_LOGIC_VECTOR(4 downto 0);
		Clk	: in	STD_LOGIC;
		Rd	: in	STD_LOGIC;
		Wr	: in	STD_LOGIC;
		Din	: in	STD_LOGIC_VECTOR (15 downto 0);
		DOut	: out	STD_LOGIC_VECTOR (15 downto 0)
			);
end RAM32x16Bits;

architecture Behavioral of RAM32x16Bits is
Subtype RAM_WORD is std_logic_vector(15 downto 0) ;
Type RAM_TABLE is array ( 0 to 31) of RAM_WORD;
signal RAM: RAM_TABLE :=
	(
		x"0000",  x"0111",  x"0222",  x"0333",
		x"0444",  x"0555",  x"0666",  x"0777",
		x"0888",  x"0999",  x"0aaa",  x"0bbb",
		x"0ccc",  x"0ddd",  x"0eee",  x"0fff",
		x"1000",  x"1111",  x"1222",  x"1333",
		x"1444",  x"1555",  x"1666",  x"1777",
		x"1888",  x"1999",  x"1aaa",  x"1bbb",
		x"1ccc",  x"1ddd",  x"1eee",  x"1fff"
		);
Begin

DOut <= RAM(conv_integer ( Addr ));
process(Clk)
begin
	if (Clk='1' and Clk'event) then
		if (Wr='1') then
			RAM (conv_integer ( Addr ))<= Din;
		end if;
	end if;
end process;
end Behavioral;
```

۲-۸. پیاده سازی فلیپ فلاپ E

بلوک دیاگرام فلیپ فلاپ E در شکل ۸-۲ نمایش داده شده است.



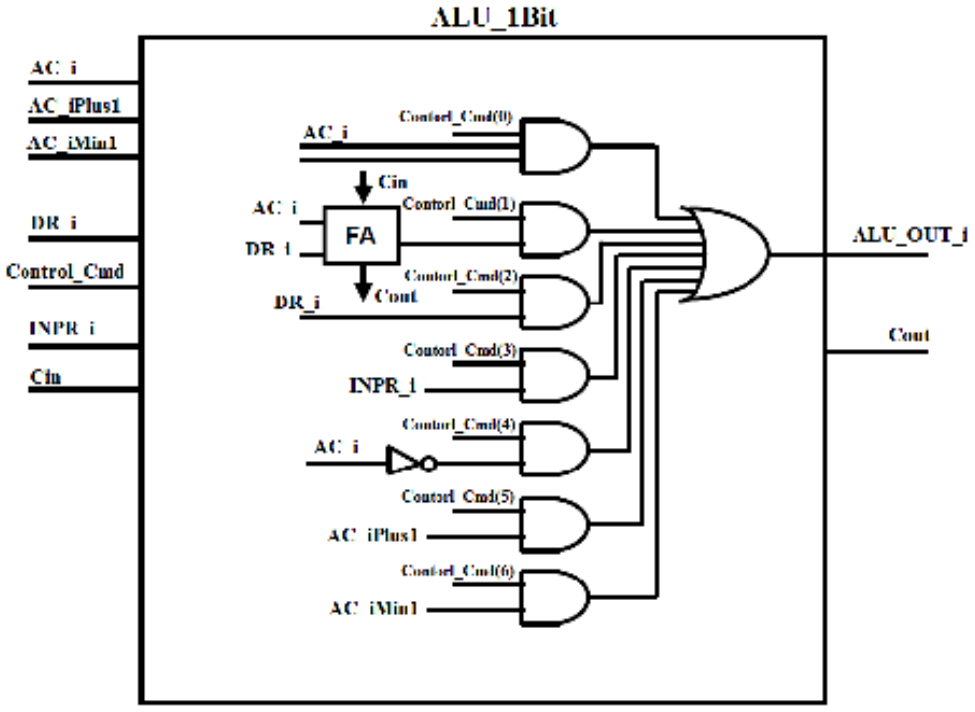
شکل ۸-۲: ورودي و خروجی هاي فليپ فلاپ E

پیاده سازی این فلیپ فلاپ مشابه برنامه نوشته شده در آزمایش ۷ برای پیاده‌سازی رجیستر می باشد.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity E_FF is
        Port (
                Clk     :       in STD_LOGIC;
                Rst     :       in STD_LOGIC;
                LD      :       in STD_LOGIC;
                Comp    :       in STD_LOGIC;
                Ein     :       in STD_LOGIC;
                Eout    :       out STD_LOGIC
                        );
end E_FF;

architecture Behavioral of E_FF is

signal Eout_sig: STD_LOGIC;
begin
        Eout <= Eout_sig;
        process(Clk)
        begin
                if(Clk='1' and Clk'event)then
                        if(Rst='1')then
                                Eout_sig<= '0';
                        elsif(LD='1')then
                                Dout_sig<= Din;
                        elsif(Comp='1')then
                                Eout_sig<= not Eout_sig;
                        end if;
                end if;
        end process;
end Behavioral;
```

هدف نهایی پروژه؛ پیاده سازی واحد ALU

برای پیاده سازی ALU بهتر است به روش ساختاری عمل کرد برای این منظور ابتدا یک ALU_1Bit طراحی می شود سپس با Port Map کردن ۱۶ عدد از آن، یک ALU شانزده بیتی طراحی می کنیم. در شکل۸-۳ بلوک دیاگرام داخلی ALU یک بیتی نمایش داده شده است.



شکل ۸-۳: پورتها و بلوک دیاگرام داخلی واحد ALU یک بیتی

۸-۴. مفهوم TestBench

برای شبیه سازی و تست یک برنامه VHDL، باید ورودیهای مورد نیاز برای آن را تولید کرد. برنامه ISIM، در محیط شبیه سازی خود یکسری سیگنالهای از قبل تعریف شده ای دارد که می توان از آنها برای شبیه سازی استفاده کرد ولی اگر بخواهیم شبیه سازی مدار را با انعطاف پذیری بیشتری انجام دهیم، باید یک برنامه VHDL نوشت که ورودیهای مورد نیاز برای تست قطعه مد نظرمان را تولید کند. به این برنامه VHDL که مخصوص شبیه سازی نوشته می شود، TestBench گفته می‌شود. TestBench هیچگونه پورت ورودی یا خروجی ندارد و در واقع Entity آن خالی است و در قسمت

Architecture باید کامپوننت مربوط به برنامه ای که می خواهیم تست کنیم را معرفی کنیم. سپس سیگنالهای ورودی مورد نیاز را تعریف کنیم و بعد از کلمه کلیدی Begin، برنامه خود برای تولید سیگنالها را بنویسیم.

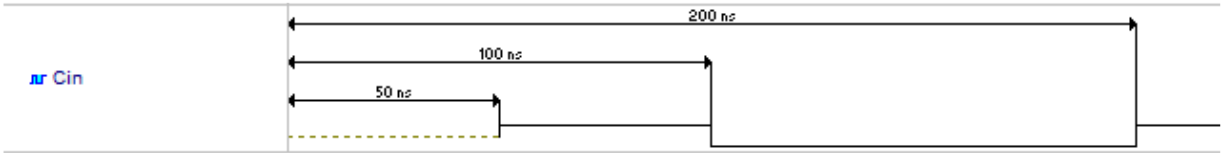
به عنوان مثال، در ادامه یک برنامه TestBench برای FullAdder آورده شده است.

```
library ieee;
use ieee.STD_LOGIC_UNSIGNED.all;
use ieee.std_logic_1164.all;
entity fulladder_tb is
end fulladder_tb;
architecture TB_ARCHITECTURE of fulladder_tb is
component fulladder
	port(
		a	:	in STD_LOGIC;
		b	:	in STD_LOGIC;
		Cin	:	in STD_LOGIC;
		Sum	:	out STD_LOGIC;
		Cout	:	out STD_LOGIC	);
end component;
	signal a	:	STD_LOGIC:='0';
	signal b	:	STD_LOGIC:='0';
	signal Cin	:	STD_LOGIC;
	signal Sum	:	STD_LOGIC;
	signal Cout	:	STD_LOGIC;
begin
	UUT : FullAdder
		port map (
			a => a,
			b => b,
			Cin => Cin,
			Sum => Sum,
			Cout => Cout
		);
	a <= not a after 10ns;
	b <= not b after 20ns;
	Cin <= '1' after 50 ns,	'0' after 100 ns,	'1' after 200ns;
end TB_ARCHITECTURE;
```

توضیحات برنامه

سیگنالهای a و b در برنامه TestBench، مقدار دهی اولیه شده اند(در هنگام تعریف این دو سیگنال) دستور a<= not a after 10ns; باعث می شود که سیگنال a هر ۱۰ نانو ثانیه مکمل شود، بنابراین سیگنال a معادل یک کلاک ۵۰ مگاهرتز خواهد بود (چرا؟!!!)

سیگنال Cin به این صورت مقداردهی شده است که بعد از ۵۰ نانو ثانیه (از لحظه شروع شبیه سازی)، مقدار آن برابر با ۱ شود بعد از ۱۰۰ نانوثانیه (از لحظه شروع شبیه سازی) مقدار آن صفر شود و نهایتا بعد از ۲۰۰ نانوثانیه (از لحظه شروع شبیه سازی)، مقدار آن ۱ شود و تا بینهایت ۱ بماند. در شکل۸-۴ شکل موج تولید شده برای سیگنال Cin نمایش داده شده است.

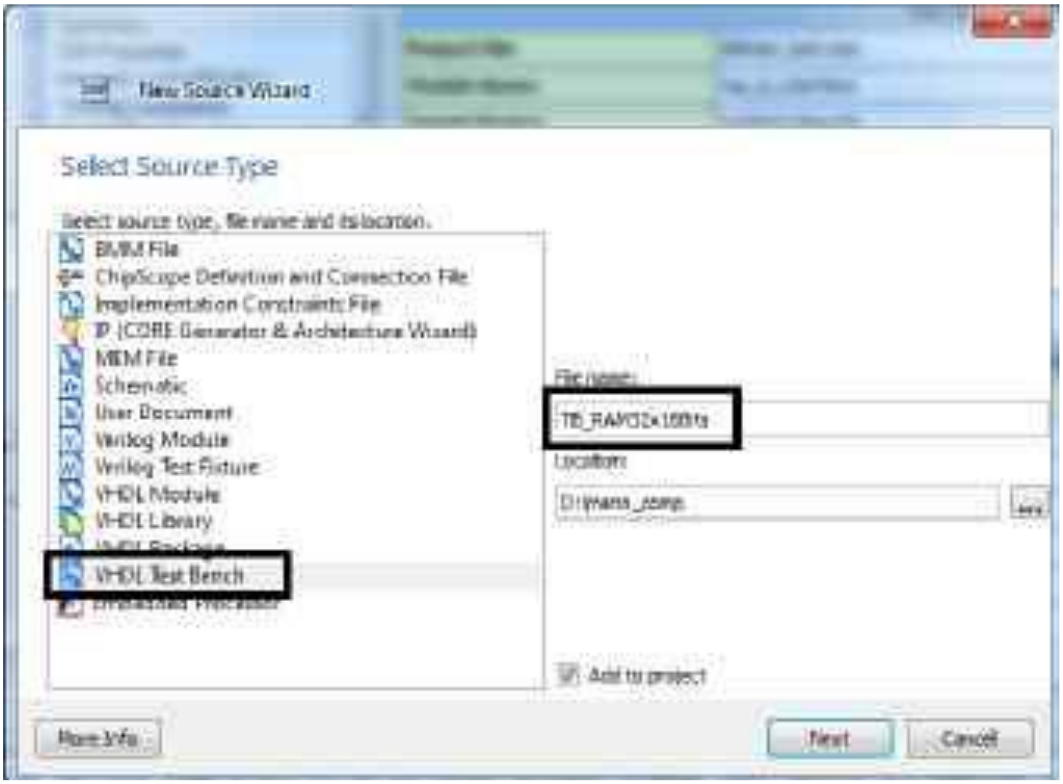


شکل ۸-٤: نتیجه دستور Cin <= '1' after 50 ns,'0' after 100 ns, '1' after 200ns

اگر وارد محیط شبیه سازی شوید و فایل testBench را برای شبیه سازی انتخاب کنید دیگر نیازی به استفاده از گزینه Simulators برای مقداردهی سیگنالها نیست و مستقیما می توانید دکمه شروع شبیه سازی را بزنید.

نکته پیاده‌سازی: دستور after قابل سنتز شدن روی FPGA نیست و فقط برای شبیه سازی استفاده می شوند. نرم افزارهای سنتز کننده از دستورهای ایجاد تاخیر صرف نظر می کنند. در صورت نیاز به تولید تاخیر در سخت افزار باید آن را با استفاده از شمارنده ایجاد کرد (در واقع تاخیر می تواند به صورت مضاربی از پریود کلاک باشد).

نرم افزار ISE این امکان را به کاربران می دهد تا چهارچوب برنامه TestBench را به صورت اتوماتیک تولید کند. برای ایجاد TestBench کافی است در پنجره پروژه، کلیک راست نموده و گزینه New Source… را انتخاب کنید سپس در پنجره ظاهر شده گزینه VHDL TestBench را انتخاب کنید و نامی برای این فایل انتخاب کنید(معمولا یک پسوند TB_ به ابتدا یا انتهای نام فایل اصلی اضافه می شود). در شکل۸-۵ نحوه ایجاد یک TestBench نمایش داده شده است.



شکل ۸-۵: نحوه ایجاد یک TestBench

۸-۵. گزارش بخش سوم پروژه

۱- با توجه به شکل ۸-۳، برنامه ALU_1Bit را بنویسید سپس با استفاده از آن برنامه ALU_16Bits را بنویسید.

۲- برنامه Mano_DataPath را که در بخش قبل نوشتید را کامل کنید و بلوکهای حافظه، ALU_16Bits و E_FF را به آن اضافه کنید.

❖ تمامی فایلهای پروژه‌ها را به همراه یک گزارش کار در یک پوشه به اسم خود قراد دهید و بعد از فشرده‌سازی ارسال نمایید.