طراحی خودکار مدارهای دیجیتال
جلسه ۴– آشنایی با Verilog (بخش ۳)

روح الله دیانت

دانشگاه قم- دانشکده فنی- گروه مهندسی کامپیوتر

خلاصه مطالب قبلی

- پیاده‌سازی گیت‌های پایه
- پیاده‌سازی مدارهای ساده ترکیبی: مدار شامل and و or، mux21 و mux41
- پیاده‌سازی سلسله‌مراتبی و امکان استفاده از ماژول‌های طراحی شده قبلی در پیاده‌سازی ماژول‌های دیگر.
- آشنایی با مفاهیم شبیه‌سازی، سنتز و برنامه‌ریزی FPGA . انجام شبیه‌سازی با Modelsim، سنتز و برنامه‌ریزی با ISE
- تعریف آرایه‌ای از خطوط و آرایه‌ای از ماژول‌ها

فهرست مطالب این جلسه

- در این جلسه با بحث معماری رفتاری (Behavioral architecture) آشنا می‌شویم که یکی از مفاهیم بسیار مهم در Verilog می‌باشد.

آشنایی با مفهوم معماری رفتاری

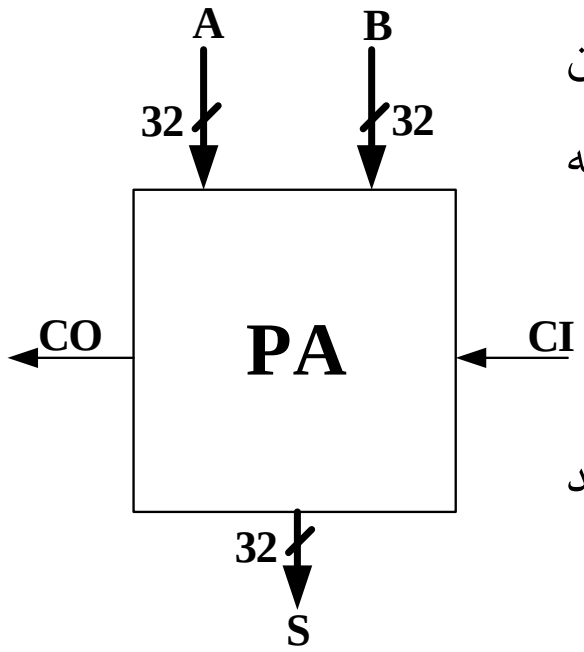
- در تمام طرح‌ها و برنامه‌هایی که تاکنون به زبان Verilog در این درس، نوشته شده است، یک وجه مشترک وجود دارد: در همه آنها، ابتدا ساختار (شکل داخلی مدار)، مشخص می‌شد و سپس، برنامه‌نویسی Verilog بر مبنای این ساختار، انجام می‌شد.
- به این نوع برنامه‌نویسی، **معماری ساختاری (Structural architecture)** اطلاق می‌گردد.
- در نقطه مقابل معماری ساختاری، روش دیگر برنامه نویسی وجود دارد که از آن، تحت عنوان معماری رفتاری یاد می‌شود.
- در معماری رفتاری، لازم نیست ساختار داخلی مدار، از ابتدا مشخص باشد. بلکه صرفا کافی است ورودی‌ها و خروجی‌ها و رفتار (ارتباط بین خروجی‌ها و ورودی‌ها)، معلوم باشد. در چنین وضعیتی، Verilog، خود، یک ساختار داخلی مناسب برای طرح را پیدا می‌نماید و طرح را بدین‌ترتیب، پیاده‌سازی می‌کند.

پیاده‌سازی جمع‌کننده موازی به صورت رفتاری

- برای فهم بهتر موارد گفته شده، به مثال زیر توجه نمایید.

- **مثال ۱:** طراحی جمع‌کننده موازی ۳۲ بیتی جلسه قبل، به صورت ساختاری طراحی شد. یعنی ابتدا ساختار داخلی مدار طراحی شده و سپس، برنامه Verilog با توجه به آن، نوشته شد. در مثال این جلسه، طراحی جمع‌کننده ۳۲ بیتی را با استفاده از مفهوم معماری رفتاری، انجام می‌دهیم.

پیاده‌سازی جمع‌کننده موازی به صورت رفتاری(ادامه)



- در معماری رفتاری لازم است ورودی، خروجی و رفتار مدار (نحوه ارتباط بین خروجی و ورودی)، مشخص باشد. برای جمع‌کننده موازی ۳۲ بیتی، این موارد به صورت زیر، مشخص است.
  - ورودی‌ها و خروجی‌ها در شکل زیر مشخص شده‌اند.
  - رفتار مدار (ارتباط بین خروجی‌ها و ورودی‌ها): خروجی S برابر A+B می‌باشد و CO هم نقلی خروجی این جمع می‌باشد.

- در برنامه صفحه بعد، پیاده‌سازی این جمع‌کننده با روش معماری رفتاری، انجام گرفته است.

پیاده‌سازی جمع‌کننده موازی به صورت رفتاری (ادامه)

- در Verilog، استفاده از اپراتور + منجر به این می‌شود که یک جمع‌کننده به مدار اضافه شود.
- در Verilog امکاناتی برای برنامه‌نویس به منظور پیاده‌سازی طرح‌ها به صورت رفتاری، ارائه شده است. یکی از این امکانات، وجود اپراتورهای مختلف مانند +، −، * و.... می‌باشد.
- برخلاف اپراتورهای موجود در یک زبان نرم‌افزاری، بعضی اپراتورهای موجود در یک زبان سخت‌افزاری مانند verilog، تعبیر فیزیکی و سخت‌افزاری دارد. مثل اپراتور + یک سخت‌افزار جمع به مدار اضافه می‌کند.

```
module PA_B(S,CO,A,B,CI);
input [32:1] A,B;
input CI;
output [32:1] S;
output CO;
assign {CO,S}=A+B;
endmodule
```

پیاده‌سازی جمع‌کننده موازی به صورت رفتاری (ادامه)

- دستور `assign` برای اختصاص مقدار به خطوط مدار استفاده می‌شود.
- نحوه اختصاص مقدار در اینجا مانند زبان‌های نرم‌افزاری نیست که عددی اختصاص داده شود. بلکه، در اینجا یک اتصال فیزیکی برقرار می‌شود. مثلا دستور `assign x=y;` خط `y` را به خط `x` به صورت فیزیکی وصل می‌کند.
- در مثال جمع‌کننده موازی دستور `A+B`، یک سخت‌افزار جمع‌کننده به مدار اضافه می‌کند که دارای سه ورودی `A`، `B` و `CI` است. این سخت‌افزار دارای دو خروجی «حاصل جمع» و «نقلی خروجی» می‌باشد. دستور `assign {CO,S}=A+B+CI`، خط `S` را به حاصل جمع و خط `CO` را به نقلی خروجی، وصل می‌کند.

```
module PA_B(S,CO,A,B,CI);
input [32:1] A,B;
input CI;
output [32:1] S;
output CO;
assign {CO,S}=A+B+CI;
endmodule
```

چند نکته در خصوص معماری رفتاری

- **نکته** (با کمک این نکته، درک بیشتری از مفهوم سخت‌افزاری دستور `assign` خواهیم داشت): فرض کنید در بخشی از یک برنامه Verilog ابتدا نوشته باشیم `assign a=0;` و سپس، در قسمتی دیگر، `assign a=1;` نوشته شده باشد. اگر این برنامه شبیه‌سازی شود متوجه می‌شویم که مقدار a برابر با × (نامشخص) شده است.

- علت این مسأله آن است که دستور `assign a=0;` ، به صورت سخت افزاری ولتاژ سطح صفر را به a وصل می‌کند. دستور `assign a=1;` که بعد نوشته شده است، تعبیرش به این شکل نیست که ولتاژ صفر برداشته شود و ولتاژ یک قرار گیرد. بلکه این دستور می‌گوید، در کنار ولتاژ صفر اعمال شده، یک ولتاژ سطح یک نیز اعمال شود. در یک مدار واقعی، این امر، موجب اعمال جریان بسیار زیاد به سیم و سوختن آن می‌گردد. در Verilog این امر، به صورت × شدن مقدار a ، مدل گردیده است.

چند نکته در خصوص معماری رفتاری (ادامه)

- آشنایی با بعضی اپراتورهای موجود در Verilog (همه این اپراتورها، تعبیر سخت افزاری دارند).
  - اپراتورهای محاسباتی) **+**، **-**، *****، **/** و **%** (باقی‌مانده تقسیم).
  - اپراتورهای منطقی) **&** (and)، **~&**(nand)، **|** (or)، **~|** (nor)، **^** (xor)، **^~** و یا **~^** (xnor).
  - اپراتورهای مقایسه‌ای) **<**، **>**، **==** و **~=**
  - اپراتور شرطی) **? :**
- **سؤال:** اپراتورهای مقایسه‌ای و شرطی، چگونه می‌توانند تعبیر فیزیکی و سخت‌افزاری داشته باشند؟ پاسخ به این سؤال در صفحات بعد داده خواهد شد.

چند مثال دیگر از معماری رفتاری و کار با اپراتورها در Verilog

❖ **مثال ۲** (در این مثال، یاد می‌گیریم که چگونه اپراتورهای مقایسه‌ای و یا شرطی دارای تعبیر سخت‌افزاری هستند). به دستور زیر توجه کنید.

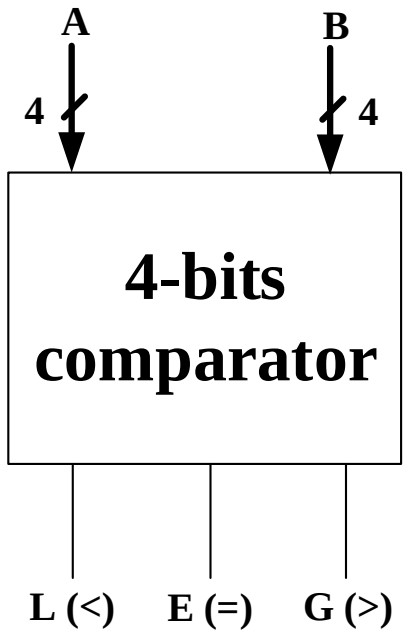
```
assign c=(a<b)?d:~d;
```

❖ در Verilog اپراتور ?: برای انجام عملیات شرطی استفاده می‌شود. دستور بالا می‌گوید اگر a<b باشد c برابر با d بشود و گرنه برابر d′.

❖ نکته بسیار مهم آن است که این دستور، تعبیر سخت‌افزاری دارد. ابزار سنتز، در پاسخ به این دستور، مدار c=a′bd+abd′+ab′d′+a′b′d′ را ایجاد می‌کند. چون اگر ab=01 باشد به علت کوچک تر بودن a از b باید d به c وصل شود که در مدار ذکر شده این اتفاق می‌افتد. اگر ab=11 باشد در مدار، c=d′ می‌شود و.... یعنی این مدار، فرایند خواسته شده در دستور assign را پیاده می‌کند.

چند مثال دیگر از معماری رفتاری و کار با اپراتورها در Verilog

- **مثال ۲:** مطلوبست پیاده‌سازی یک مقایسه‌کننده ۴ بیتی .
- پیاده‌سازی ساختاری مقایسه‌کننده ۴ بیتی، دشوار است. لذا، از روش طراحی رفتاری استفاده می‌کنیم.



- در بلاک روبرو:
  - اگر A<B باشد آنگاه LEG=100.
  - اگر A>B باشد آنگاه LEG=001.
  - اگر A=B باشد آنگاه LEG=010.

چند مثال دیگر از معماری رفتاری و کار با اپراتورها در Verilog (ادامه)

- با استفاده از معماری رفتاری، پیاده‌سازی مقایسه‌گر ۴ بیتی به آسانی انجام شد.

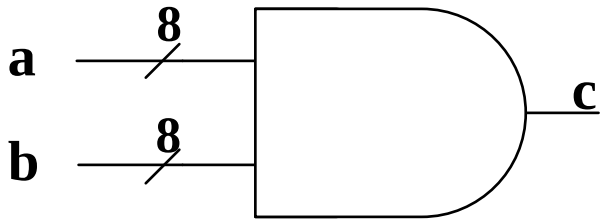
```
module comparator4
(L,E,G,A,B);
input [4:1] A,B;
output L,E,G;
assign L=(A<B)? 1 : 0;
assign E=(A==B)?1:0;
assign G=(A>B)?1:0;
endmodule
```

```
module comparator4_test;
reg [4:1] A,B;
wire L,E,G;
comparator4 comp(L,E,G,A,B);
initial begin
A=4'b0000;  B=4'b0001;
#100;
A=4'b1000;  B=4'b0001;
#100;
A=4'b0000;  B=4'b0000;
end
endmodule
```

چند مثال دیگر از معماری رفتاری و کار با اپراتورها در Verilog (ادامه)

❖**مثال ۳:** برنامه‌ای برای پیاده‌سازی and ۸ بیتی بنویسید.

برنامه نوشته در زیر، بر مبنای معماری رفتاری نوشته شده است. اگر می‌خواستیم این برنامه را به روش عماری ساختاری بنویسی، ۸ بار احضار ماژول and را لازم داشت.




```
module and8 (c,a,b);
input  [7:0] a,b;
output [7:0] c;
assign c=a&b;
endmodule
```

چند مثال دیگر از معماری رفتاری و کار با اپراتورها در Verilog (ادامه)

❖**مثال ۴:** مطلوب است طراحی یک mux 2*1 به روش رفتاری.

```
module mux21 (O,I0,I1,S);
input  I0,I1,S;
output O;
assign O=(S==0)?I0:I1;
endmodule
```

چند مثال دیگر از معماری رفتاری و کار با اپراتورها در Verilog (ادامه)

- **نکته: (اولویت اپراتورها در Verilog):** مانند سایر زبان‌ها، میان اپراتورهای موجود در Verilog، اولویت بندی در اجرا وجود دارد.
- در جدول صفحه بعد، تقدم اپراتورها از بالاترین اولویت به پایین، ذکر شده است.
- ترتیب تقدم و تأخر اپراتورهای موجود در یک سطح، از چپ به راست عبارت می‌باشد.

چند مثال دیگر از معماری رفتاری و کار با اپراتورها در Verilog (ادامه)

انتخاب بیت	[]
پرانتز	( )
نقیض منطقی نقیض بیتی	! ~
علامت مثبت علامت منفی	+ -
اتصال	{ }
ضرب تقسیم باقی‌مانده	* / %
جمع دودویی تفریق دودویی	+ -
شیفت به چپ شیفت به راست	<< >>

چند مثال دیگر از معماری رفتاری و کار با اپراتورها در Verilog (ادامه)

بزرگ‌تر، بزرگترمساوی، کوچکتر، کوچکترمساوی	> >= < <=
تساوی منطقی ، عدم تساوی منطقی	== !=
And بیتی	&
Xor بیتی xnor بیتی	^ ^~ or ~^
OR بیتی	|
And منطقی	&&
OR منطقی	||
شرطی سه گانه	?:

چند مثال دیگر از معماری رفتاری و کار با اپراتورها در Verilog (ادامه)

❖**مثال ۵:** مطلوب است طراحی یک mux 4*1 به روش رفتاری.

در این مثال با استفاده از ? : پیاده‌سازی mux41 انجام شده است. همان گونه که ملاحظه می‌شود، دستور assign پیاده کننده mux41 نسبتا پیچیده شده است. در جلسه بعد، روش‌های بهتری در معماری رفتاری را فرا می‌گیریم که نوشتن چنین برنامه‌هایی را آسان‌تر می‌کند.

```
module mux41 (O,I,S);
input  [3:0] I;
input  [1:0] S;
output O;
assign O=(S==2'b00)?I[0]:((S==2'b01)?I[1]:((S==2'b10)?I[2]:I[3]));
endmodule
```

پایان