دانشگاه تهران - دانشکدهٔ فنی
گروه مهندسی برق و کامپیوتر

خودآموز زبان توصیف سخت‌افزاری Verilog

تهیه کننده : سعید صفری

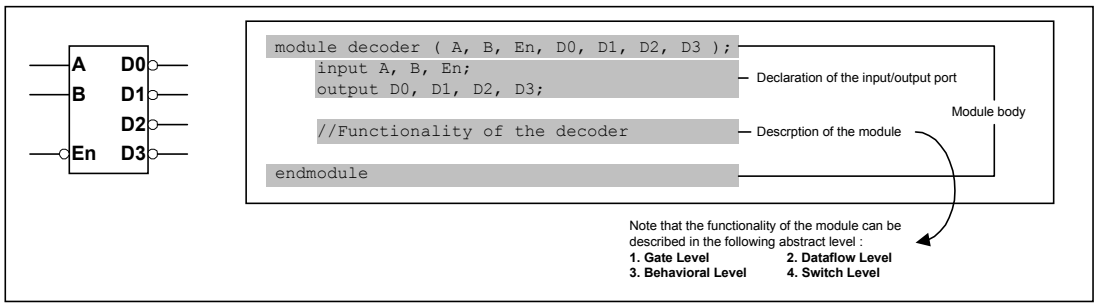
اسفند ۷۹

۱ مفاهیم اولیه

۱-۱ ماجول

ماجول بلوک پایهٔ Verilog است. یک ماجول می‌تواند یک عنصر یا مجموعه‌ای از بلوک‌های سطح پایین‌تر باشد. بنابراین ماجول عملکرد مورد نظر را برای بلوکهای سطح بالاتر فراهم می‌کند، اما پیاده‌سازی داخل آنرا پنهان می‌کند. شکل ۱-۱ نحوهٔ تعریف ماجول را با یک مثال بیان می‌کند. در زبان Verilog می‌توانیم یک ماجول را در چهار سطح مختلف تجرید بیان کنیم :

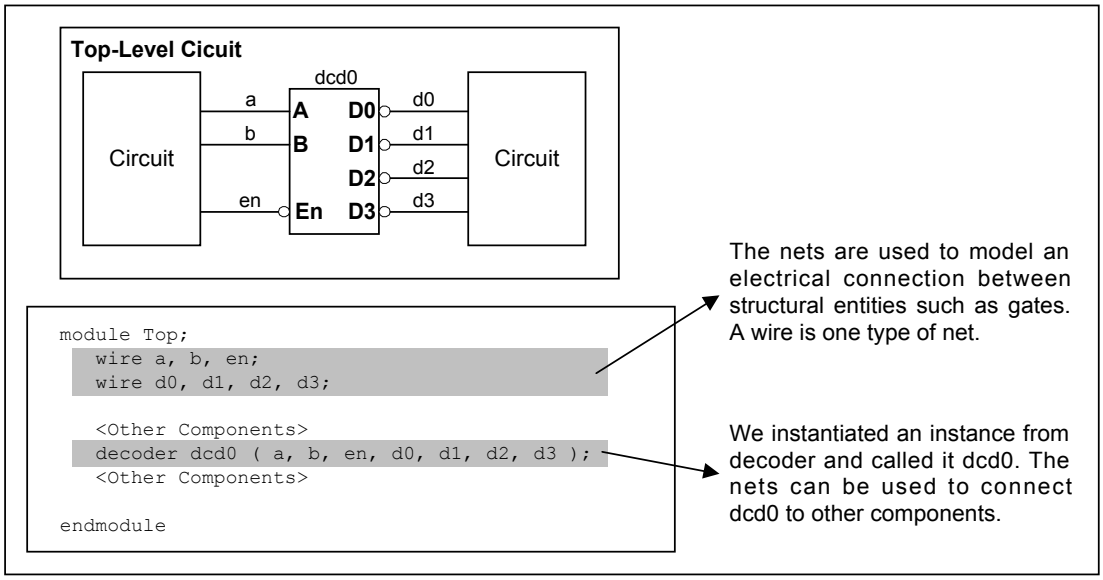
- **سطح گیت** : در این سطح ماجول بصورت گیتهای منطقی و اتصالات بین آنها بیان می‌شود.
- **سطح جریان داده (Dataflow)** : در این سطح ماجول بوسیلهٔ مشخص کردن نحوهٔ جریان اطلاعات بین رجیسترها و نوع پردازشی که روی آنها صورت می‌گیرد، بیان می‌شود.
- **سطح رفتاری (Behavioral)** : در این سطح ماجول برحسب الگوریتم طراحی شود، بدون اینکه جزئیات طراحی پیاده‌سازی سخت‌افزاری در نظر گرفته شود.
- **سطح سوئیچ** : در این سطح ماجول بصورت سوئیچها و اتصالات بین آنها بیان می‌شود.



شکل ۱-۱- نحوهٔ تعریف ماجول

۱-۲ نمونه

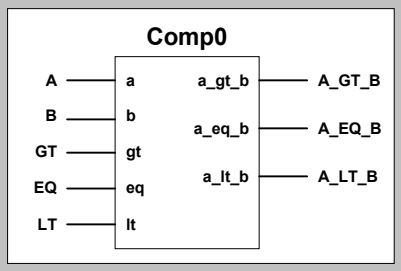
یک ماجول الگویی از یک عنصر واقعی می‌سازد، هنگامیکه از این ماجول استفاده می‌شود، Verilog یک نمونه از این الگو می‌سازد. هر عنصر دارای نام، متغیرها و پارامترهای خاص خود است. پروسهٔ ایجاد یک نمونه از الگوی یک ماجول را اصطلاحا Instantiation یا نمونه‌سازی و این عنصر را Instance یا نمونه می‌نامند. بعنوان مثال در شکل ۱-۲ نحوهٔ ساخت یک نمونه از روی ماجول دیکودر را می‌بینیم.



شکل ۱-۲- نحوهٔ ساخت نمونه از روی ماجول دیکودر

درسش!

- **در شکل زیر یک مقایسه‌گر بیتی نشان داده شده است. یک ماجول بنام Comparator تعریف کنید و سپس یک نمونه بنام Comp0 از روی آن بسازید. دقت کنید هدف فقط تعریف پورتهای مقایسه‌گر و نمونه‌سازی از روی آن است.**



مقایسه‌گر بیتی

۲ قراردادهای نحوی

قردادهای نحوی که بوسیلهٔ Verilog استفاده می‌شود، بسیار شبیه زبان برنامه‌نویسی C است. هر توکن می‌تواند توضیح، جداکننده، عدد، رشته، شناسه و یا کلمهٔ کلیدی باشد. Verilog یک زبان حساس به متن است و تمام کلمات کلیدی آن با حروف کوچک نوشته می‌شوند.

۲-۱ حروف فاصله

حروف فاصله در Verilog عبارتند از : فاصله (\b) ، tab (\t) و خط‌جدید (\n). این حروف توسط Verilog نادیده گرفته می‌شوند، مگر اینکه بعنوان جداکننده توکنها استفاده شوند و یا در یک رشته استفاده شوند.

۲-۲ توضیحات

توضیحات برای خواناتر کردن طرح بکار می‌روند و به دوصورت یک وچندخطی استفاده می‌شود. توضیحات نمی‌توانند بصورت تودرتو استفاده شوند.

```
a = b && c; // This is a single line comment
/* This is a multiple line
   comment */
```

۲-۳ اپراتورها

اپراتورها به سه دسته یگانی، دوتایی و سه‌تایی تقسیم می‌شوند و به ترتیب دارای یک، دو و سه اپرند هستند.

۲-٤ مشخصات اعداد

در Verilog اعداد به دو صورت زیر نوشته می‌شوند :

- **عدد اندازه دار** : در اینحالت Verilog را مقید می‌کنیم یک عدد را با همان اندازهٔ موردنظر ما بکار ببرد. شکل عدد اندازه‌دار بصورت <size>‘<radix><number> است. <size> به دسیمال نوشته شده و تعداد بیتهای عدد را مشخص می‌کند. <radix> مبنای عدد را مشخص می‌کند، d یا D برای مبنای ده، b یا B برای مبنای دو، o یا O برای مبنای هشت و h یا H برای مبنای شانزده بکار می‌رود. حالت پیش‌فرض مبنای ده است. برای خواناترشدن اعداد می‌توان از "_" استفاده نمود.

```
8'b1010_1110    // This is a 8 bit binary number
12'hab6         // This is a 12 bit hex number
16'D255         // This is a 12 bit decimal number
-4'd13          // This is a 4 bit negative decimal number, that stored
                // as 2's complement of 13
```

- **عدد بدون اندازه** : شکل کلی عدد بدون اندازه بصورت ’<radix><number> است. در این حالت طول عدد به نوع پیاده‌سازی بستگی دارد ولی حداقل ۳۲ بیت است.

```
1234            // This is a 32 bit decimal number
h'62            // This is a 32 bit hex number
o'255           // This is a 32 bit octal number
```

- **اعداد منفی با قرار دادن یک علامت منفی ("-") قبل از <size> بدست می‌آیند.**
- **کرکتر ؟ و x معادلا بکار می‌روند.**
- **در Verilog دونماد برای مقادیر HiZ و نامعلوم داریم که به ترتیب عبـارتند از x, z . این مقادیر در شبیه‌سازی مدارها بسیار مورد اسـتفاده قـرار مـی‌گـیرند. در مقداردهی به یک عدد درمبنای شانزده، هشت و دو مقادیر x, z بـه‌ترتیب طولـی برابر ٤ و ٣ و ١ بیت دارند.**

```
12'h13x        // This is a 12 bit hex number, 4 LSBs unknown
4'b10??        // This is a 4 bit binary number equal to 4'b10xx
```

- **مشخص کنید که کدام یک از اعداد زیر معتبر هستند.**

659 'h83F 4AF 12'hx 5'D3 8'd-6 27_13_45

٢-٥ رشته

مجموعه‌ای است از کرکترهـا کـه بوسـیله "" محصـور شـده‌اند.

٢-٦ شناسه و کلمه کلیدی

کلمات کلیـدی شناسـه‌هایی هسـتند کـه از پیـش بـرای تعریـف سـاختار زبـان اسـتفاده شـده‌اند. کلمـات کلیدی با حـروف کـوچـک نوشـته می‌شـوند. شناسـه‌ها نامـهایی هسـتند کـه مـا بـه عنـاصر نسـبت می‌دهیـم تـا بوسیلهٔ آن بـه آنـها رجـوع کنیـم. شناسـه می‌توانـد از کرکترهـای حرفـی، _ و $ تشـکیل شـود و حتمـا بـاید بـا یک کرکتر حرفی شروع شـود. شناسـه‌ها حسـاس بـه متـن هسـتند.

٢-٧ انواع داده‌ها

٢-٧-١ مجموعه مقادیر

Verilog بـرای مدلسـازی عملکـرد سـخت‌افزارهای واقعـی از ٤ مقـــدار و ٨ ســطح قــدرت اســتفاده می‌کند. مجموعـهٔ مقـادیر در جـدول ١-١ و سـطوح قـدرت در جـدول ١-٢ آمـده اسـت.

Value Level	Condition in Hardware Circuits
0	Logic Zero, False Condition
1	Logic One, True Condition
x	Unknown value
z	High Impedance, Floating State

جدول ۱-۱- مجموعهٔ مقادیر

Strength Level	Type	Degree
supply	Driving	strongest
strong	Driving	
pull	Driving	
large	Storage	
weak	Driving	
medium	Storage	
small	Storage	
highz	High Impedance	Weakest

جدول ۱-۲- سطوح قدرت

۲-۷-۲ Net

برای برقرار کردن ارتباط بین اجزاء سخت‌افزاری بکار می‌رود. درست مانند مدارهای واقعی، net دارای مقداری است که بوسیلهٔ خروجی عنصر متصل به آن روی آن درایو می‌شود. net در Verilog توسط کلمهٔ کلیدی wire تعریف می‌شود و مقدار پیش فرض آن z است.

```
wire a; // Declare net a
wire b=1'b0    // Net b is fixed to logic value 0 at declaration
```

- نحوهٔ برخورد با z در ورودیهای گیتها و عبارات دقیقا مانند x است، فقط ترانزیستورهای MOS حالت z را ازخود عبور می‌دهند.
- سطوح قدرت برای حل تصادم بین دو درایور مختلف در نظر گرفته می‌شود.
- اگر دو سیگنال نامساوی با سطوح قدرت مختلف یک سیم را درایو کنند، سیگنال قویتر برنده می‌شود.
- اگر دو سیگنال نامساوی با سطوح قدرت مساوی یک سیم را درایو کنند، حاصل نامعلوم می‌شود.

۲-۷-۳ Register

برای ذخیرهٔ اطلاعات بکار می‌رود. رجیستر تا وقتی مقدار جدیدی روی آن نوشته نشده مقدار خود را نگاه می‌دارد. برخلاف net ، register به درایور نیاز ندارد. register در Verilog توسط کلمهٔ کلیدی reg تعریف می‌شود و مقدار پیش‌فرض آن x است.

```
reg reset;      // Declare a variable that can be hold its value
```

۲-۷-٤ Vector

انواع داده‌ای reg و wire می‌توانند بصورت بردار تعریف شوند. شکل کلی تعریف بردار بصورت زیر است :

<vector_type> [MSB : LSB] <vector_name>

```
wire [31:0] BusA;      // Declare a bus that has 32 bit width
reg [0:40] Vir_Add;    // virtual address 41 bits wide
```

۲-۷-۵ انواع داده‌ای صحیح، حقیقی و زمان

- **صحیح** : یک نوع دادهٔ رجیستر همه منظوره است که برای پردازش مقادیر صحیح استفاده می‌شود. تفاوت این نوع با نوع reg دراینستکه در نوع reg داده‌ها بصورت بدون علامت در نظر گرفته می‌شوند، ولی در نوع صحیح داده‌ها بصورت علامتدار در نظر گرفته می‌شوند. نوع داده‌ای صحیح توسط کلمهٔ کلیدی integer تعریف می‌شود و طول آن بستگی به پیاده‌سازی دارد ولی حداقل ۳۲ بیت درنظر گرفته می‌شود.

```
integer counter;      // General purpose variable used as a counter
```

- **حقیقی** : یک نوع دادهٔ رجیستر همه منظوره است که برای پردازش مقادیر صحیح استفاده می‌شود. نوع داده‌ای صحیح توسط کلمهٔ کلیدی integer تعریف می‌شود.

```
real delta;           // Declare a real variable called delta
```

- **زمان** : در Verilog یک نوع داده‌ای خاص برای ذخیره کردن زمان شبیه‌سازی استفاده می‌شود. نوع داده‌ای زمان بوسیله کلمه کلیدی time تعریف می‌شود. طول متغیر از نوع زمان به پیاده‌سازی بستگی دارد ولی حداقل ٦٤ بیت است.

```
time sim_time;        // Define a time variable sim_time
```

۲-۸ آرایه

در Verilog می‌توان آرایه‌ای از نوعهای داده‌ای reg, integer, time و یا آرایه‌ای از بردارهایی از این نوعها تعریف کرد. شکل کلی تعریف آرایه عبارتست از :

<array_type> <array_name> [#first_element : #last_element]

```
integer count[0:7];   // An array of 8 integer
```

- در Verilog آرایه‌های چندبعدی نداریم.
- تفاوت آرایه و بردار در اینستکه آرایه از چند عنصر تشکیل شده است که هریک از آنها می‌توانند یک یا چند بیت طول داشته باشند، درحالیکه بردار یک عنصر است که طولی برابر n دارد.

۲-۹ حافظه

در Verilog حافظه را بصورت آرایه‌ای از رجیسترها تعریف می‌کنیم.

```
reg membit[0:1023];                 // 1K x 1bit memory
reg [7:0] membyte[0:1023];          // 1K x 8bit memory
```

۲-۱۰ پارامتر

می‌توان در یک ماجول اعداد ثابتی را بصورت پارامتر تعریف نمود و از آنها استفاده نمود، این امر توسط کلمه کلیدی parameter انجام می‌شود.

```
parameter port_id = 5;       // Define a constant port_id
```

۲-۱۱ رشته

رشته‌ها می‌توانند در یک reg ذخیره شوند. طول متغیر reg باید به اندازه کافی بزرگ باشد تا بتواند رشته را نگاه دارد. هر کرکتر در ۸ بیت ذخیره می‌شود.

```
reg [8*18:1] s_val;   // Define a variable with 18 bytes
```

۲-۱۲ task های سیستم

در Verilog تعدادی task به منظور نمایش اطلاعات روی صفحه، مانیتورکردن مقادیر، خاتمهٔ شبیه‌سازی و... فراهم شده است. کلیهٔ task ها بصورت $<task_name> بکار می‌روند.

- **نمایش اطلاعات** : نمایش اطلاعات توسط $display انجام می‌شود و دارای شکل کلی $display(format, p1, p2, ..., pn); است، که در آن pi ها می‌توانند نام متغیر، نام سیگنال و یا رشته باشند. توسط format می‌توان قالب نمایش اطلاعات را بصورت دلخواه تعیین نمود. برای این منظور یک سری قالب از پیش تعریف شده است که لیست آنها در جدول ۱-۳ آمده است.

Format	Display
%d or %D	Display variable in decimal
%b or %B	Display variable in binary
%s or %S	Display string
%h or %H	Display variable in hex
%c or %C	Display ASCII character
%v or %V	Display strength
%o or %O	Display variable in octal
%t or %T	Display in current time format
%e or %E	Display real number in scientific format
%f or %F	Display real number in decimal format
%g or %G	Display real number in scientific or decimal format, whichever is shorter

جدول ۱-۳- لیست مشخصات قالبها

```
$display("At time %t virtual address is %h", $time, vir_Adr );
```

- **مانیتورکردن اطلاعات** : مانیتورکردن اطلاعات توسط $monitor انجام می‌شود و دارای شکل کلی $monitor(format, p1, p2, ..., pn); است، که در آن pi ها می‌توانند نام متغیر، نام سیگنال و یا رشته باشند. تفاوت نمایش اطلاعات با مانیتورکردن اطلاعات در اینستکه، $display با هربار فراخوانی یک مرتبه مقادیر پارامترهایش را نشان می‌دهد ولی $monitor بطور دائم مقادیر پارامترهایش را مانیتور می‌کند و به محض تغییر یکی از پارامترهای آن، مقادیر کلیهٔ پارامترهایش را نشان می‌دهد. باید توجه داشت که در هرلحظه فقط یک $monitor می‌تواند فعال باشد، بنابراین اگر چندین دستور $monitor داشته باشیم، فقط آخرین دستور $monitor فعال است. توسط دو task به نامهای $monitoron و $monitoroff می‌توان عملیات مانیتور کردن اطلاعات را به ترتیب فعال و غیرفعال نمود.
- **توقف و خاتمهٔ شبیه‌سازی** : توسط $stop می‌توان عملیات شبیه‌سازی را متوقف نمود و آنرا در مد interactive قرارداد. این مد برای عیب‌یابی بکار می‌رود. توسط $finish می‌توان عملیات شبیه‌سازی را خاتمه داد.

۲-۱۳ راهنمای کامپایلر

در Verilog راهنمای کامپایلر دارای شکل کلی `<keyword> است. دو نوع راهنمای کامپایلر که مورد استفاده بیشتری دارند، عبارتند از :

- `define که برای تعریف ماکرو بکار می‌رود.
- `include که برای الحاق یک فایل Verilog به فایل جاری بکار می‌رود.

```
`define WORD_SIZE 32          // Used as `WORD_SIZE in the code
`include "header.v"           // Include the file header.v
```

۳ ماجول

قبلا نحوهٔ تعریف و نمونه‌سازی ماجول را دیدیم، در اینجا بصورت دقیقتر اجزا ماجول را مورد بررسی قرار می‌دهیم. در شکل ۱-۳ اجزا ماجول مشخص شده است. قسمتهایی که بصورت زیرخط‌دار نوشته شده‌اند، در تعریف ماجول ضروری و سایر قسمتها اختیاری است. قسمتهایی که بدنهٔ ماجول را تشکیل می‌دهند، می‌توانند با هر ترتیبی در تعریف ماجول استفاده شوند.

```
module module_name ( port list );
    port declarations (if ports present)
    parameters (optional)

Declaration  of wires, regs and other variables

Data flow statemnet (assign)

Instantiation of lower level module

always and initial blocks, all behavioral
statements go in these blocks

tasks and functions

endmodule
```

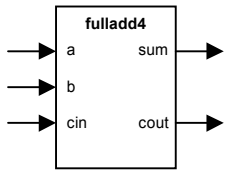
شکل ۱-۳- اجزاء تشکیل دهنده ماجول

٤ پورت

پورتهای یک ماجول، واسط ارتباطی ماجول با جهان خارج است.

٤-۱ لیست پورتها

به هنگام تعریف ماجول دیدیم که لیست پورتها (درصورت وجود) در جلوی نام ماجول معرفی می‌شوند. بعنوان مثال در شکل ۱-٤ یک جمع‌کنندهٔ ٤ بیتی و نحوهٔ تعریف پورتهای آن نشان داده شده است.




```
module fulladd4 ( sum, cout, a, b, cin );
```

شکل ۱-٤- جمع‌کننده ٤ بیتی و نحوه تعریف پورتهای آن

٤-۲ تعریف پورتها

تمام پورتهای یک ماجول باید تعریف شوند. این تعریف طبق جدول ۱-٤ صورت می‌گیرد.

Verilog Keyword	Type of port
input	Input Prt
output	Output Prt
inout	Bidirectional Prt

جدول ۱-٤- تعریف پورتها

• کلیهٔ پورتها بطور ضمنی بصورت wire تعریف می‌شوند، مگر اینکه بخواهیم یک خروجی مقدارش را نگاه‌دارد که در اینصورت آنرا از نوع reg تعریف می‌کنیم.

بعنوان مثال در مورد جمع‌کنندهٔ شکل ۱-٤ تعریف پورتها بصورت زیر است.

```
module fulladd4( sum, cout, a, b, cin );
   output [3:0] sum;
   output cout;

   input [3:0] a, b;
   input cin;

   < Module Body >

endmodule
```

٤-۳ قوانین اتصال پورتها

به هنگام اتصال پورتهای یک ماجول به جهان خارج باید به نکاتی توجه داشت :

- پورتهای ورودی ماجول باید از نوع net باشند و این پورتها می‌توانند به متغیرهایی از نوع reg و یا net درجهان خارج متصل شوند.
- پورتهای خروجی ماجول می‌توانند از نوع reg و یا net باشند و این پورتها باید به متغیرهایی از نوع net درجهان خارج متصل شوند.

- پورتهای دوسویهٔ ماجول باید از نوع net باشند و این پورتها می‌توانند به متغیرهایی از نوع reg و یا net درجهان خارج متصل شوند.
- پورتهای ماجول و متغیرهای خارجی متصل به آنها باید از نظر طول منطبق باشند.

در Verilog به دو صورت می‌توان ارتباط پورتها را با جهان خارج برقرار نمود :

۱- **اتصال ترتیبی** : در این روش به هنگام نمونه‌سازی از یک ماجول، متغیرهای متصل به پورتها را دقیقا به همان ترتیبی که در تعریف ماجول آمده‌اند، بیاوریم. در این روش اگر بخواهیم یک پورت خروجی به جایی متصل نباشد کافیست جای آن را در لیست خالی بگذاریم.

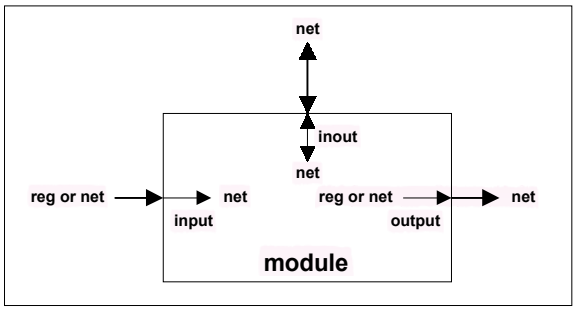
۲- **اتصال از طریق نام** : در این روش برای اتصال هر پورت از قالب زیر استفاده می‌کنیم :

.port_name(external_signal_name)

از این روش وقتی استفاده می‌کنیم که تعداد پورتها زیاد باشد و بخاطر سپردن ترتیب آنها دشوار باشد. در این روش اگر بخواهیم یک پورت خروجی به جایی متصل نباشد کافیست نام آن را نیاوریم.

```
//-- First Method ------------------------------------------
fulladd4 fa0( s, co, x, y, ci );
fulladd4 fa0( s,   , x, y, ci );

//-- Second Method -----------------------------------------
fulladd4 fa0( .sum(s), .cout(co), .cin(ci), .a(x), .b(y) );
fulladd4 fa0( .sum(s), .cin(ci), .a(x), .b(y) );
```

در شکل ۱-۵ قوانین اتصال پورتها نمایش داده شده است.



شکل ۱-۵- قوانین اتصال پورتها

۵ مدلسازی در سطح گیت

در مدلسازی در سطح گیت مدار را بصورت مجموعه‌ای از گیتهای پایه که به یکدیگر متصل شده‌اند بیان می‌کنیم. برای این منظور باید انواع گیتهای پایه‌ای که در Verilog وجود دارند را معرفی کنیم.

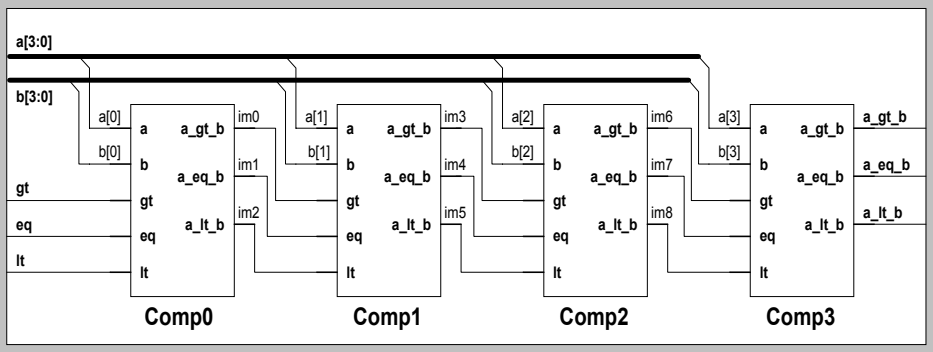
۵-۱ گیتهای پایه

۵-۱-۱ گیتهای and/or

جدول صحت گیتهای پایهٔ and/or در شکل ۱-۶ آمده است.

پرسش ۴

• با اتصال چهار مقایسه‌گر یک بیتی مطابق شکل زیر ، یک ماجول مقایسه‌گر چهاربیتی بنام **Comparator4** بسازید. فرض کنید که ماجول مقایسه‌گر یک بیتی بنام **Comparator** موجود است. سیگنالهای داخلی مورد نیاز را از **im0** تا **im8** نامگذاری کنید.



and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

nand	0	1	x	z
0	1	1	1	1
1	1	0	x	x
x	1	x	x	x
z	1	x	x	x

nor	0	1	x	z
0	1	0	x	x
1	0	0	0	0
x	x	0	x	x
z	x	0	x	x

xnor	0	1	x	z
0	1	0	x	x
1	0	1	x	x
x	x	x	x	x
z	x	x	x	x

شکل ۱-۶- جدول صحت گیتهای پایهٔ and/or

```
and a1( out, in1, in2 );          // and gate with two inputs
or ( z, i1, i2 );                 // or gate with two inputs
xor ( z1, i1, i2, i3, i4 );       // xor gate with four inputs
nand nd( o1, i1, i2, i3 );        // nand gate with three inputs
nor ( out, a, b, c );             // nor gate with three inputs
xnor ( y, a, b, c, d );           // xnor gate with four inputs
```

نکته

• پورت اول این گیتها، پورت خروجی و بقیهٔ پورتها ورودی هستند. به این ترتیب می‌توان گیتهایی با تعداد ورودیهای دلخواه داشت.

• به هنگام نمونه‌سازی (استفادهٔ) از این گیتها، می‌توان نام نمونه را ذکر نکرد (این امر در مورد تمام المانهای از پیش ساخته شدهٔ **Verilog** صادق است).

۵-۱-۲ گیتهای buf/not

جدول صحت گیتهای پایهٔ buf/not در شکل ۱-۷ آمده است.

buf	out
0	0
1	1
x	x
z	x

	ctrl			
bufif0	0	1	x	z
0	0	z	L	L
1	1	z	H	H
x	x	z	x	x
z	x	z	x	x

	ctrl			
bufif1	0	1	x	z
0	z	0	L	L
1	z	1	H	H
x	z	x	x	x
z	z	x	x	x

not	out
0	1
1	0
x	x
z	x

	ctrl			
notif0	0	1	x	z
0	1	z	H	H
1	0	z	L	L
x	x	z	x	x
z	x	z	x	x

	ctrl			
notif1	0	1	x	z
0	z	1	H	H
1	z	0	L	L
x	z	x	x	x
z	z	x	x	x

شکل ۱-۷- جدول صحت گیتهای پایهٔ buf/not

نکته

- **پورت آخر گیتهای not, buf ، پورت ورودی و بقیهٔ پورتها خروجی هستند. به این ترتیب می‌توان گیتهایی با تعداد خروجیهای دلخواه داشت.**
- **در مورد گیتهای دیگر پورت اول خروجی، پورت بعدی ورودی و پورت سوم کنترل است.**
- **L به معنای 0 یا z و H به معنای 1 یا z می‌باشد.**

```
buf b1 ( o1, o2, i1 );              // buffer with 2 outputs
notif1 ( z_bar, in, ctrl );         // not with output control
```

برای طراحی مدار در سطح گیت، ابتدا باید آنرا بصورت مجموعه‌ایی از گیتهای پایه درآورد، سپس با تعریف net های مورد نیاز این گیتهای پایه را به یکدیگر متصل نمود، به این عمل اصطلاحا Wiring یا سیم‌بندی گفته می‌شود. اکنون با ذکر چند مثال نحوهٔ طراحی مدار در سطح گیت را بیان می‌کنیم.

مثال۱ : یک مالتی‌پلکسر ٤ به ۱ طراحی کنید.

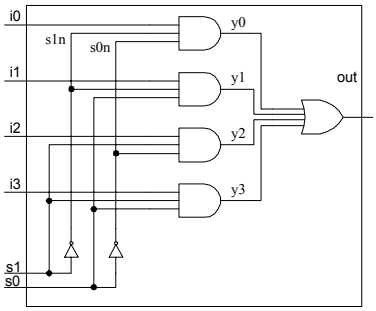
حل : طرح شماتیک مدار در سطح گیت و کد Verilog مربوط به آن در شکل ۱-۸ آمده است.

مثال۲ : یک مقایسه‌کننده تک بیتی قابل توسعه طراحی کنید.

حل : طرح شماتیک مدار در سطح گیت و کد Verilog مربوط به آن در شکل ۱-۹ آمده است.

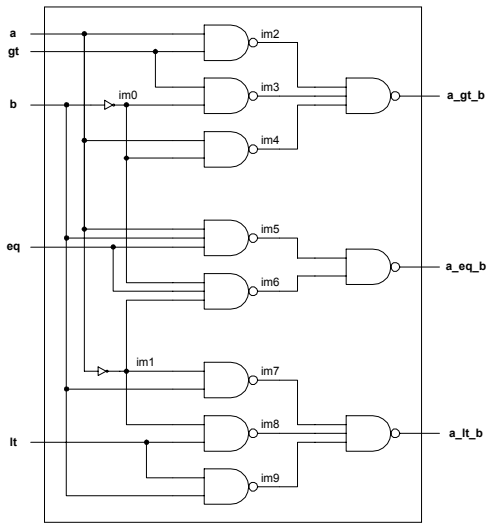
درستش کن

- **با استفاده از گیتهای nand یک فلیپ‌فلاپ نوع D که با لبهٔ بالارونده Clk عمل می‌کند بسازید. ورودیهای مدار D, Clk و خروجیهای مدار q, q_bar می‌باشد.**




```
// Module 4_to_1 multiplexer.
module mux4_to_1 ( out, io, i1, i2, i3, s1, s0 );
   output out;
   input io, i1, i2, i3, s1, s0;

   // Internal wire declaration
   wire s0n, s1n, y0, y1, y2, y2;

   // Gate instantiations
   not ( s1n, s1 );
   not ( s0n, s0 );

   and ( y0, i0, s1n, s0n );
   and ( y1, i1, s1n, s0  );
   and ( y2, i2, s1 , s1n );
   and ( y3, i3, s1 , s0  );

   or ( out, y0, y1, y2, y3 );

endmodule
```

شکل ۱-۸- نقشهٔ شماتیک مالتی‌پلکسر ٤ به ۱ در سطح گیت و کد Verilog مربوطه




```
//Module 1-bit comparator

module Comparator( a_gt_b, a_eq_b, a_lt_b,
                   a, b, gt, eq, lt );

   output a_gt_b, a_eq_b, a_lt_b;
   input  a, b, gt, eq, lt;

   // Internal wire declaration
   wire im0,im1,im2,im3,im4,im5,im6,im7,im8,im9;

   // Gate instantiation
   not ( im0, b );
   not ( im1, a );
   nand ( im2, a, gt );
   nand ( im3, gt, im0 );
   nand ( im4, a, im0 );
   nand ( im5, a, b, eq );
   nand ( im6, im0, eq, im1 );
   nand ( im7, im1, b );
   nand ( im8, im1, lt );
   nand ( im9, lt, b );
   nand ( a_gt_b, im2, im3, im4 );
   nand ( a_eq_b, im5, im6 );
   nand ( a_lt_b, im7, im8, im9 );

endmodule
```

شکل ۱-۹- نقشهٔ شماتیک مقایسه‌گر تک بیتی قابل توسعه در سطح گیت و کد Verilog مربوطه

۲-۵ تاخیر گیتها

تاکنون گیتهایی کـه در نظـر گرفتیـم حـالت ایـده‌آل داشـتند، یعنـی بـه محـض تغییـر ورودی بـدون هیـچ تـاخیری خروجـی تغییـر پیـدا می‌کـرد. ولـی در مدارهـای عملـی وضعیـت بـه اینصـورت نیسـت. تعییـن تـاخیر برای گیتها اجـازه می‌دهـد شبیه‌سـازی مدارهـا حـالت واقعی‌تـری بـه‌خود بگـیرد. مقـادیر تـاخیر در Verilog بـا علامت # شـــروع می‌شـود.

۵-۲-۱ تاخیرهای Turn-Off, Fall, Rise

- هرگونه تغییر در خروجی گیت از 0, x, z به 1 با تاخیر Rise انجام می‌شود.
- هرگونه تغییر در خروجی گیت از 1, x, z به 0 با تاخیر Fall انجام می‌شود.
- هرگونه تغییر در خروجی گیت به z با تاخیر Turn-Off انجام می‌شود.
- اگر خروجی گیت به x تغییر وضعیت بدهد، مینیموم این سه تاخیر در نظر گرفته می‌شود.
- به هنگام نمونه‌سازی از گیت اگر :
- فقط یک تاخیر مشخص شود، این مقدار برای تمام تاخیرها درنظر گرفته می‌شود.
- دوتاخیر مشخص شود، به ترتیب برای تاخیرهای Rise و Fall در نظر گرفته می‌شوند و تاخیر Turn-Off برابر مینیموم این دو مقدار در نظر گرفته می‌شود.
- سه تاخیر مشخص شود، به ترتیب برای تاخیرهای Rise و Fall و Turn-Off در نظر گرفته می‌شوند.


```
and #(5) ( o1, i1, i2 );          // Rise=5, Fall=5, Turn-Off=5
and #(4,6) ( o1, i1, i2 );        // Rise=4, Fall=6, Turn-Off=4
and #(4,5,6) ( o1, i1, i2 );      // Rise=4, Fall=5, Turn-Off=6
```

۵-۲-۲ مقادیر Min/Typ/Max

در Verilog هر یک از تاخیرها دارای یک مقدار مینیموم، یک مقدار معمولی و یک مقدار ماکزیموم است. این تاخیرها دارای شکل کلی Min : Typ : Max هستند. انتخاب یکی از این تاخیرها در زمان اجرا صورت می‌گیرد. مثلا در بعضی شبیه‌سازها وقتی اجرا با سوئیچ maxdelay+ انجام شود، برای تمام گیتها مقدار ماکزیموم تاخیرها در نظر گرفته می‌شود.

```
// R=Rise Dealy       F=Fall Dealy        T=Turn-Off Delay

// One delay is specified
// if +mindelay, R = F = T = 4
// if +typdelay, R = F = T = 5
// if +maxdelay, R = F = T = 6
and #(4:5:6) ( o1, i1, i2 );

// Two delays are specified
// if +mindelay, R = 3, F = 5, T = min(3,5)
// if +typdelay, R = 4, F = 6, T = min(4,6)
// if +maxdelay, R = 5, F = 7, T = min(5,7)
and #( 4:5:6, 5:6:7 ) ( o1, i1, i2 );
```

```
// Three delays are specified
// if +mindelay, R = 2, F = 3, T = 4
// if +typdelay, R = 3, F = 4, T = 5
// if +maxdelay, R = 4, F = 5, T = 6
and #( 2:3:4, 3:4:5, 4:5:6 ) ( o1, i1, i2 );
```

- **توجه کنید که تاخیر گیتها فقط به هنگام شبیه‌سازی مدارها مفید هستند، و وقتی هدف سنتز مدار می‌باشد به هیچ‌وجه نباید از تاخیر در توصیف مدار استفاده شود.**

٦ مدلسازی در سطح جریان داده

در مدلسازی در سطح جریان داده به نحوهٔ انتقال اطلاعات بین ثباتها و پردازش اطلاعات اهمیت می‌دهیم. در این قسمت به چگونگی مدلسازی در سطح جریان داده در زبان Verilog و مزایای آن می‌پردازیم.

٦-١ Continuous Assignment

توسط این دستور می‌توان یک مقدار را روی یک net درایو کرد. شکل کلی این دستور بصورت زیر است :

assign <signal_strength> <delay> assignment_lists;

- **در مورد این دستور باید توجه داشت که این دستور همواره فعال است و هرگاه یکی از اپرندهای سمت راست دستور تغییر کند، کل عبارت سمت راست مجددا ارزیابی شده در متغیر سمت چپ قرار می‌گیرد.**
- **سمت چپ این دستور باید یک متغیر یا یک بردار از نوع net و یا الحاقی از این دو نوع باشد.**

```
// A dataflow description of a 2 inputs and gate
module myand( out, i1, i2 );
    output out;
    input i1, I2;
    assign out = i1 & i2;
endmodule
```

٦-٢ Implicit Continuous Assignment

در این روش بجای اینکه یک متغیر را از جنس net تعریف کنیم و سپس توسط دستور assign یک مقدار را روی آن درایو کنیم، می‌توانیم این عمل را در هنگام تعریف متغیر net انجام دهیم.

```
wire out;
assign out = i1 & i2;
// are same as
wire out = i1 & i2;
```

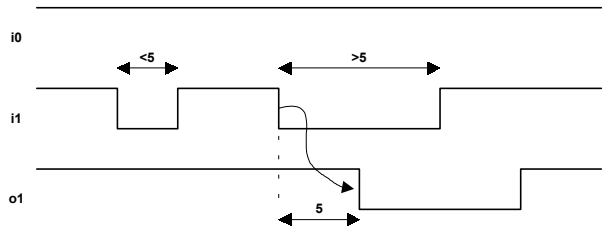
۶-۳ تاخیرها

همانطورکه در سطح گیت دیدیم تاخیرها برای این استفاده می‌شوند که عملکرد مدار به واقعیت نزدیکتر باشد. در سطح جریان داده نیز می‌توان برای assignment ها تاخیر مشخص نمود.

۶-۳-۱ تاخیر با قاعده

در اینحالت یک تاخیر را پس از assign و قبل از net می‌آوریم. هر تغییری که در یکی از سیگنالهای سمت راست رخ دهد، باعث می‌شود پس از گذشت زمان تاخیر، عبارت سمت راست مجددا ارزیابی شود و سپس در متغیر سمت چپ قرار گیرد. به این ترتیب تاخیری که در اینجا داریم، از نوع Inertial است و این امر باعث می‌شود که پالسهایی با عرض کمتر از مقدار تاخیر مشخص شده به خروجی منتشر نشود. این امر در شکل ۱-۱۰ نمایش داده شده است.

```
module myand ( o1, i1, i2 );
    output o1;
    input i1, i2;

    assign #5 o1 = i1 & i2;

endmodule
```



شکل ۱-۱۰- تاخیر انتشار Inertial

۶-۳-۲ تاخیر ضمنی

دراینحالت تاخیر و assignment روی یک net به هنگام تعریف آن net مشخص می‌شود.

```
wire out;
assign #5 out = i1 & i2;
// are same as
wire #5 out = i1 & i2;
```

۶-۳-۳ تاخیر به هنگام تعریف net

دراینحالت به هنگام تعریف net تاخیر مورد نظر را برای آن مشخص می‌کنیم. از این پس هرتغییری که روی این net انجام شود، با تاخیر مشخص شده اعمال می‌شود. باید توجه داشت که این امر در مورد مدلسازی در سطح گیت نیز قابل استفاده است.

```
wire #5 out;
assign out = i1 & i2;
```

٤-٦ عبارات، اپراتورها و اپرندها

- یک عبارت از ترکیب اپرندها و اپراتورها ساخته می‌شود.
- یک اپرند می‌تواند یکی از انواع داده‌ای باشد که در بخش ۱-۲ به آن اشاره شد.
- اپراتورها روی اپرندها عملیاتی را انجام می‌دهند تا نتیجهٔ مطلوب بدست آید. لیست اپراتورهای موجود در Verilog در جدول ۱-۵ آمده است.

Operator Type	Operator Symbol	Operation Performed	Number of Operands
Arithmetic	*	multiply	2
	/	division	2
	+	add	2
	-	subtract	2
	%	modulus	2
Logical	!	logical negation	1
	&&	logical and	2
	||	logical or	2
Relational	>	greater than	2
	<	less than	2
	>=	greater than or equal	2
	<=	less than or equal	2
Equality	==	equality	2
	!=	inequality	2
	===	case equality	2
	!==	case inequality	2
Bitwise	~	bitwise negation	1
	&	bitwise and	2
	|	bitwise or	2
	^	bitwise xor	2
	~^ or ^~	bitwise xnor	2
Reduction	&	reduction and	1
	~&	reduction nand	1
	|	reduction or	1
	~|	reduction nor	1
	^	reduction xor	1
	~^ or ^~	reduction xnor	1
Shift	>>	right shift	2
	<<	left shift	2
Reduction	{}	concatenation	any number
	{ {} }	replication	any number
Conditional	? :	conditional	3

جدول ۱-۵- لیست اپراتورهای Verilog برای مدلسازی در سطح جریان داده

- در مورد اپراتورهای حسابی باید به موارد زیر توجه داشت :
- اگر هردو اپرندش صحیح باشد، خارج‌قسمت را برمی‌گرداند.
- اگرهریک ازاپرندها دارای بیت x باشد، نتیجهٔ عملیات x خواهد بود.
- % باقیماندهٔ تقسیم را برمی‌گرداند و علامت حاصل برابر علامت اپرند اول است.
- بهتر است اعداد منفی را درعبارات بصورت اعداد صحیح بکار برد، زیرا در غیراینصورت به مکمل ۲ تبدیل می‌شوند که ممکن است باعث بروز نتایج غیرمنتظره شوند.

```
A = 4'b0011;   B = 4'b0100;     D = 6;   E = 4;
A*B     // Evaluated to 4'b1100
D/E     // Evaluated to 1
A+B     // Evaluated to 4'b0111
B-A     // Evaluated to 4'b0001
//-------------------------------------------------------
in1 = 4'b101x;   in2 = 4'b1010;
sum = in1 + in2;     // Evaluated to 4'x

//-------------------------------------------------------
-7 % 2  // Evaluated to -1, take sign of the first operand
7 % -2  // Evaluated to +1, take sign of the first operand
```

- در مورد اپراتورهای منطقی باید به موارد زیر توجه داشت :
- نتیجهٔ اپراتورهای منطقی یک بیت است : 0 نادرست، 1 درست، x نامعلوم.
- اگر اپرند 0 باشد معادل نادرست، اگر 1 باشد معادل درست و اگر x باشد معادل نامعلوم ارزیابی می‌شود.

```
A = 3;   B = 0;
A && B  // Evaluated to 0 ( False )
A || B  // Evaluated to 1 ( True  )
!A      // Evaluated to 0 ( False )

//-------------------------------------------------------
A = 2'bx0;   B = 2'b10;
A && B  // Evaluated to x ( Unknown )
```

- در مورد اپراتورالحاق باید به موارد زیر توجه داشت :
- اپرندها حتما باید اعداد اندازه‌دار باشند تا Verilog قادر به محاسبهٔ اندازهٔ نتیجه باشد.
- اپرندها می‌توانند net ، reg ، برداری از net ، reg یا اعداد اندازه‌دار باشند.

```
A = 1'b1;   B = 2'b00;   D = 2'b10;   C = 3'b110;
y = { B, C };                        // Result y is 4'b0010
y = { A, B ,C, D, 3'b001 };          // Result y is 11'b1_00_10_110_001
y = { A, b[0], C[1] };               // Result y is 3'b101
```

- در مورد اپراتورتکرار باید به موارد زیر توجه داشت :
- یک ثابت تکرار مشخص می‌کند که چندبار عدد داخل {} باید تکرار شود.

```
A = 1'b1;   B = 2'b00;   D = 2'b10;

y = { 4{A} };                // Result y is 4'b1111
y = { 4{A}, 2{B} };          // Result y is 8'b11110000
y = { 4{A}, 2{B}, C };       // Result y is 10'b1111000010
```

- در مورد اپراتور شرطی باید به موارد زیر توجه داشت :
- قالب استفاده از اپراتور شرطی بصورت زیر است :

<condition> ? true_exp : false_exp;

- عبارت **condition** ترزیابی می‌شود، اگر نتیجه عبارت **x** باشد، هردو عبارت **true_exp, false_exp** محاسبه شده و بیت به بیت با هم مقایسه می‌شوند، اگر بیتها باهم متفاوت بودند **x** و درغیراینصورت همان بیت برگردانده می‌شود.

```
// Models functionality of a tri state buffer
assign addr_bus = drive_enable ? addr_out : 32'bz;
```

- در مورد اپراتورهای تساوی باید به موارد زیر توجه داشت :
- نتیجهٔ **a==b** برابر **0** یا **1** یا **x** است. اگر یک بیت از یکی از اپرندها **x** یا **z** باشد، نتیجه **x** می‌شود.
- نتیجهٔ **a!=b** برابر **0** یا **1** یا **x** است. اگر یک بیت از یکی از اپرندها **x** یا **z** باشد، نتیجه **x** می‌شود.
- نتیجهٔ **a===b** برابر **0** یا **1** است. **a** و **b** بیت به بیت با هم مقایسه می‌شوند.
- نتیجهٔ **a!==b** برابر **0** یا **1** است. **a** و **b** بیت به بیت با هم مقایسه می‌شوند.

```
A = 3;   B = 3;
X = 4'b1010;   Y = 4'b1101;
Z = 4'b1xzz;   M = 4'b1xzz;    N = 4'b1xxx;

A == B         // Result is logical 0
A != B         // Result is logical 1
X == Z         // Result is x
Z === M        // Result is logical 1
Z === N        // Result is logical 0
M !== M        // Result is logical 1
```

- **در مورد اپراتورهای کاهشی (reduction) باید به موارد زیر توجه داشت :**
- **این اپراتورها دارای یک اپرند هستند و عملیات بیتی مشخص شده را روی تک‌تک اعضای بردار اپرند آن انجام داده و یک بیت را بعنوان نتیجه برمی‌گرداند.**

```
x = 4'b1010;
&x             // Equivalent to 1 & 0 & 1 & 0, Result is 1'b0
|x             // Equivalent to 1 | 0 | 1 | 0, Result is 1'b0
^x             // Equivalent to 1 ^ 0 ^ 1 ^ 0, Result is 1'b0
```

حال سعی می‌کنیم با ذکر چند مثال به نحوهٔ مدلسازی مدارهای دیجیتال در سطح جریان داده بپردازیم.

مثال۱ : یک مالتی‌پلکسر ٤ به ١ طراحی کنید.

```
// Dataflow model of a 4-to-1 multiplexer

module mux4_to_1 ( out, i0, i1, i2, i3, s1, s0 );
    output out;
    input i0, i1, i2, i3, s1, s0;

    // Use nested conditional operator
    assign out = s1 ? ( s0 ? i3 : i2 ) : ( s0 ? i1 : i0 );

endmodule
```

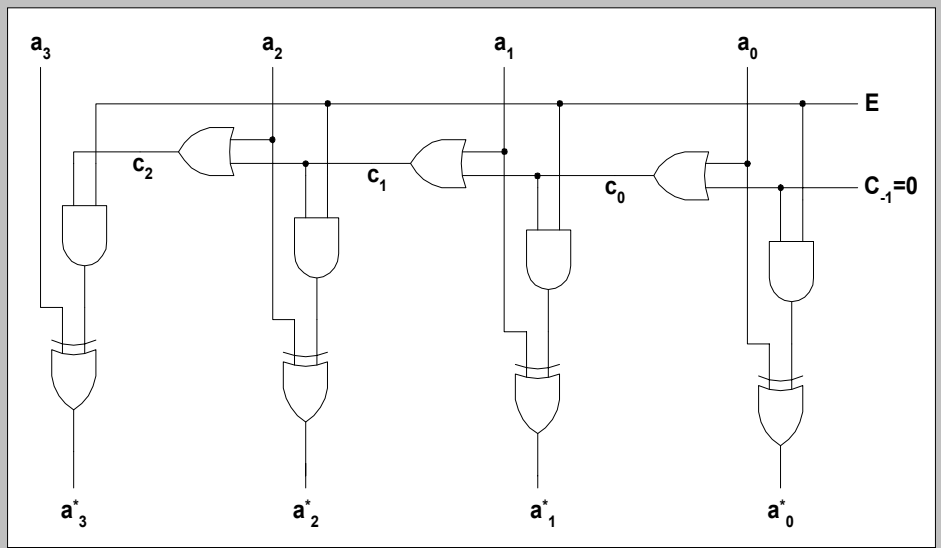
مثال۲ : یک جمع‌کنندهٔ ٤ بیتی طراحی کنید.

```
// Dataflow model of a 4-bit full adder

module fulladd4 ( sum, c_out, a, b, c_in );
    output [3:0] sum;
    output c_out;
    input [3:0] a, b;
    input c_in;

    // Specify the function of a 4-bit full adder
    assign { c_out, sum } = a + b + c_in;

endmodule
```

مثال۳ : یک مقایسه‌کننده تک بیتی قابل توسعه طراحی کنید.

```
// Dataflow model of a cascadable 1-bit comparator

module Comparator ( a_gt_b, a_eq_b, a_lt_b, a, b, gt, eq, lt );
    output a_gt_b, a_eq_b, a_lt_b;
    input a, b, gt, eq, lt;

    // Specify the boolean function of a 1-bit comparator
    assign a_gt_b = ( a & gt ) | ( ~b & gt ) | ( a & ~b );
    assign a_eq_b = ( a & b & eq ) | ( ~a & ~b & eq );
    assign a_lt_b = ( ~a & lt ) | ( b & lt ) | ( ~a & b );

endmodule
```

- در شکل زیر طرح شماتیک مداری که مکمل ۲ یک عدد باینری ٤ بیتی را محاسبه می‌کند نشان داده شده است. این مدار را در سطح جریان داده توصیف کنید.



پرسش ۵

۷ مدلسازی در سطح رفتاری

۷-۱ بلوکهای ساخت‌یافته

در Verilog دو بلوک ساخت‌یافته وجود دارد، initial ,always . این دستورات پایهٔ مدلسازی در سطح رفتاری هستند و تمام قسمتهای توصیف رفتاری مدار در این بلوکهای ساخت‌یافته قرار می‌گیرند. این بلوکها دارای ساختار ترتیبی هستند.

۷-۱-۱ بلوک initial

بلوک initial در زمان 0 شروع شده و فقط یکبار در شروع شبیه‌سازی اجرا می‌شود. چنانچه بخواهیم درون بلوک initial چندین دستور داشته باشیم، باید بوسیلهٔ begin end آنها را بلوک کنیم. چنانچه چندین بلوک initial داشته باشیم، تمام بلوکها در زمان 0 بصورت همزمان اجرا می‌شوند و هر بلوک مستقل از سایر بلوکها خاتمه می‌یابد. اگر قبل از یک دستور تاخیری مشخص شود، آن دستور پس از تاخیر مشخص شده از زمان فعلی شبیه‌سازی اجرا می‌شود.

```
module stimulus;

    reg a, b, x, y, m;

    initial
        m = 1'b0;

    initial
    begin
        #10 x = 1'b0;
        #10 y = 1'b1;
    end



    initial
    begin
        #5 a = 1'b1;
        #10 b = 1'b0;
    end

    initial
        #50 $finish;

endmodule

//      Time            Statement executed
//       0                  m = 1'b0;
//       5                  a = 1'b1;
//       10                 x = 1'b0;
//       30                 b = 1'b0;
//       35                 y = 1'b1;
//       50                 $finish;
```

۲-۱-۷ بلوک always

بلوک always در زمان 0 شروع شده و تمام دستورات درون آن بطور پیوسته اجرا می‌شوند (مانند یک حلقه). این دستورات برای مدلسازی یک بلوک از اعمال که متوالیا در یک مدار انجام می‌شوند، بکار می‌رود.

```
module clock_gen;

   reg clk;

   initial
      clk = 1'b0;

   // Toggle clk every half_cycle ( Period = 20 )
   always
      #10 clk = ~clk;

   initial
      #1000 $finish;

endmodule
```

۲-۷ Procedural Assignment

این دستور مقدار یک متغیر reg ، صحیح، حقیقی یا زمان را تغییر می‌دهد. مقدار جدید در متغیر باقی می‌ماند تا هنگامیکه یک دستور دیگر مقدار آن را تغییر دهد. سمت چپ این عبارت می‌تواند یکی از موارد زیر باشد :

- یک متغیر reg ، صحیح، حقیقی، زمان یا عناصر حافظه
- یک Bit-Select از این متغیرها
- یک Part-Select از این متغیرها
- الحاقی از موارد فوق

۱-۲-۷ Blocking Assignment

این دستورات به همان ترتیبی که مشخص شده‌اند اجرا می‌شوند، یعنی بدون کامل شدن اجرای یک دستور، دستور بعدی اجرا نمی‌شود. اپراتوری که برای این امر بکار می‌رود = است.

```
initial
begin

   // These statements are executed at time 0 sequentially
   x = 0;   y = 1;   z = 1;
   count = 0;
   reg_a = 16'h0000;  reg_b = reg_a;

   // This statement is executed at time 15
   #15 reg_a[2] = 1'b1;
```

```
    // These statements are executed at time 25 sequentially
    #10 reg_b[15:13] = { x, y, z };
    count = count + 1;

end
```

۷-۲-۲ Nonblocking Assignment

در این روش کلیهٔ دستورات برای اجرا زمانبندی می‌شوند، بدون اینکه منتظر کامل شدن اجرای یک دستور باشیم. اپراتوری که برای این امر بکار می‌رود <= است.

- **شبیه‌ساز یک دستور Nonblocking را برای اجرا زمانبندی می‌کند، سپس به دستور بعدی درون بلوک می‌پردازد، بدون اینکه منتظر کامل شدن اجرای دستور قبلی شود.**
- **دستورات Nonblocking می‌توانند بصورت مؤثری همزمانی انتقال اطلاعات را مدل کنند، زیرا نتیجهٔ نهائی به ترتیب اجرای دستورات وابسته نیست.**

```
initial
begin

    x = 0;    y = 1;    z = 1;
    count = 0;
    reg_a = 16'h0000;   reg_b = reg_a;

    // This statement is scheduled to execute after 15 time units
    reg_a[2] <= #15 1'b1;

    // This statement is scheduled to execute after 10 time units
    reg_b[15:13] <= #10 { x, y, z };

    // This statement is scheduled to execute without any delay
    count <= count + 1;

end
```

۷-۳ کنترل زمان

در Verilog چنانچه دستورات کنترل زمان موجود نباشد، شبیه‌سازی انجام نمی‌شود. در Verilog سه نوع کنترل زمان وجود دارد :

۷-۳-۱ کنترل زمان مبتنی بر تاخیر

در این روش یک عبارت، فاصلهٔ زمانی بین رسیدن به یک دستور تا اجرای آن را مشخص می‌کند. دو نوع کنترل زمان مبتنی بر تاخیر موجود است.

- **کنترل تاخیر با قاعده** : یک تاخیر غیر صفر در سمت چپ دستور آورده می‌شود. دراینحالت عبارت سمت راست پس از گذشت زمان تاخیر، محاسبه شده درون عبارت سمت چپ قرار می‌گیرد.

```
initial
begin
   x = 0;
   #10 y = 1;
   #(4:5:6) q=0;
end
```

- **کنترل تاخیر درون دستور** : یک تاخیر غیر صفر در سمت راست اپراتور assignment آورده می‌شود. دراینحالت عبارت سمت راست در زمان فعلی محاسبه شده، پس از گذشت زمان تاخیر درون عبارت سمت چپ قرار می‌گیرد.

```
initial
begin
   x = 0;    z = 0;
   y = #5 x + z;
end
// Is equivalent to
initial
begin
   x = 0;    z = 0;     temp = x + z;
   #5 y = temp;
end
```

۷-۳-۲ کنترل زمان مبتنی بر رویداد

یک رویداد به معنای تغییر مقدار یک reg یا net است. چهارنوع کنترل زمان مبتنی بر رویداد وجود دارد.

- **کنترل رویداد با قاعده** : علامت @ برای مشخص کردن کنترل رویداد استفاده می‌شود. دستورات می‌توانند با تغییر مقدار یک سیگنال، با لبهٔ بالارونده یا پایین روندهٔ یک سیگنال اجرا شوند. لبهٔ بالارونده به معنی یکی از تغییرات $0 \to 1,\ x, z; x \to 1; z \to 1$ و لبهٔ بالارونده به معنی یکی از تغییرات $1 \to 0,\ x, z; x \to 0; z \to 0$ می‌باشد.

```
@(clock) q = d;              // Triggered with any change in clock
@(posedge clock) q = d;      // Triggered positive edge of clock
@(negedge clock) q = d;      // Triggered negative edge of clock
q = @(posedge clock) d;      // d is evaluated immediately and
                             // assigned to q at negative edge of clock
```

- **کنترل رویداد با نام** : Verilog این امکان را برای ما فراهم ساخته است که یک رویداد را تعریف کنیم و در موقع لزوم آنرا تریگر کنیم. تعریف رویداد با کلمهٔ کلیدی event و تریگر کردن آن با -> انجام می‌شود.

```
event rec_data;

always @(posedge clock)
begin
   if( last_data_packet ) ->rec_data;
end

always @(rec_data)
data_buf = { data[0] , data[1], data[2], data[3] };
```

- **کنترل چند رویداد** : گاهی اوقات چند سیگنال داریم که تغییر در یکی از آنها سبب تریگر شدن اجرای یک مجموعه از دستورات می‌شود. این امر توسط or کردن رویدادها یا سیگنالها انجام می‌شود. لیست رویدادا یا سیگنالها به Sensitivity List مشهور است.

```
always @(posedge clock or reset)
begin
   if( reset ) q = 0;
   else q = d;
end
```

۷-۳-۳ کنترل حساس به سطح

Verilog دارای این قابلیت است که اجرای یک دستور را تا تحقق یک شرط خاص به تعویق بیاندازیم، این امر توسط دستور wait انجام می‌شود.

```
always
   wait( count_enable ) #20 count = count + 1;
```

۷-٤ دستور شرطی

```
if( expr )
   true_st
else
   false_st;
```

۷-۵ دستور case

```
case ( expr )
   case 1 : st1;
   case 2 : st2;
   ...
   case n : stn;
   defult : def_st;
endcase
```

- **عبارت expr با تک‌تک موارد بیت به بیت مقایسه می‌شود (با در نظرگرفتن x, 1, 0).**
- **اگر طول عبارت با موارد مورد مقایسه یکسان نباشد، سمت چپ جزء کوچکتر با ۰ پر می‌شود.**
- **درصورت استفاده از casex مقادیر z, x بصورت بی‌اهمیت (d'ont care) تلقی می‌شوند یعنی در مقایسه نقشی ندارند.**
- **درصورت استفاده از casez مقدار z بصورت بی‌اهمیت (d'ont care) تلقی می‌شود یعنی در مقایسه نقشی ندارد.**

۶-۷ حلقه‌ها

```
while( expr )
   st;

//---------------------------------------
for( init; end_cond; chang_control_var )
   st;

//---------------------------------------
repeat( number_of_iteration )
   st;

//---------------------------------------
forever
   st;
```

۷-۷ Tasks and Functions

Task و تابع دارای تفاوتهایی هستند که در جدول ۱-۶ نشان داده شده است.

۱-۷-۷ Task

بوسیلهٔ task ... endtask مشخص می‌شود. معمولا در موارد زیر استفاده می‌شود.

- در پروسیجر تاخیر، زمانبندی و یا کنترل زمان وجود داشته باشد.
- پروسیجر پارامتر خروجی نداشته باشد و یا بیش از یک پارامتر خروجی داشته باشد.
- پروسیجر هیچ پارامتر ورودی نداشته باشد.

Task	Function
یک task می‌تواند توابع و task های دیگر را فعال کند.	یک تابع فقط می‌تواند توابع دیگر را فعال کند.
یک task ممکن است در زمان 0 اجرا می‌شود.	یک تابع همیشه در زمان 0 اجرا می‌شود.
یک task می‌تواند تاخیر، رویداد و یاکنترل زمان داشته باشد.	یک تابع نمی‌تواند تاخیر، رویداد و یاکنترل زمان داشته باشد.
یک task می‌تواند آرگومانهای ورودی، خروجی یا دوسویه داشته باشد.	یک تابع حداقل باید یک آرگومان ورودی داشته باشد.
یک task نمی‌تواند مقدار بازگشتی داشته باشد.	یک تابع همیشه مقدار بازگشتی دارد.

جدول ۱-٦- تفاوتهای Task و تابع

```
module operation;
   reg [15:0] A, B;
   reg [15:0] AB_AND, AB_OR, AB_XOR;

   always @(A or B) //whenever A or B changes in value
      bitwise_oper(AB_AND, AB_OR, AB_XOR, A, B);

   // Define task bitwise_oper
   task bitwise_oper;
      output [15:0] ab_and, ab_or, ab_xor;      //outputs from the task
      input [15:0] a, b;                        //inputs to the task
   begin
     #20 ab_and = a & b;
         ab_or = a | b;
         ab_xor = a ^ b;
   end
   endtask

endmodule
```

۲-۷-۷ Function

بوسیلهٔ function ... endfunction مشخص می‌شود. معمولا در موارد زیر استفاده می‌شود.

- در پروسیجر تاخیر، زمانبندی و یا کنترل زمان وجود نداشته باشد.
- پروسیجر یک پارامتر خروجی داشته باشد.
- پروسیجر حداقل یک پارامتر ورودی داشته باشد.

```
function calc_parity;
input [31:0] address;
begin

   // set the output value appropriately. Use the implicit
   // internal register calc_parity.

   calc_parity = ^address; //Return the ex-or of all address bits.

end
endfunction
```

۸-۷ طراحی مدارهای ترکیبی در سطح رفتاری

برای طراحی مدارهای ترکیبی در سطح رفتاری، باید تمام ورودی‌های مدار را در لیست حساس بدنه‌ی always ذکر کرد. به هنگام توصیف مدار باید توجه داشت که تمام شرط‌های if باید دارای else باشند تا از ایجاد مدار ترتیبی جلوگیری شود.

مثال : یک مالتی‌پلکسر ٤ به ١ طراحی کنید.

```
// Behavioral model of a 4-to-1 multiplexer

module mux4_to_1 ( out, i0, i1, i2, i3, s1, s0 );
    output out;
    reg out;
    input i0, i1, i2, i3, s1, s0;

    always @( i0 or i1 or i2 or i3 or s1 or s0 )
    begin
       case { s1, s0 }
           2'b00 : out = i0;
           2'b01 : out = i1;
           2'b10 : out = i2;
           2'b11 : out = i3;
           defualt : out = 1'bx;
       endcase
    end

endmodule
```

۹-۷ طراحی مدارهای ترتیبی در سطح رفتاری

طراحی مدارهای ترتیبی را با استفاده از چند مثال بیان می‌کنیم.

مثال ١ : یک D-FF حساس به لبه‌ی بالارونده با reset سنکرون طراحی کنید.

```
// Behavioral model of a d-ff with synchronous reset

module d_ff ( d, clk, rst, q );
    input d, clk, rst;
    output q;
    reg q;

    always @( posedge clk )
    begin
       if (rst)
           q = 1'b0;
       else
           q = d;
    end

endmodule
```

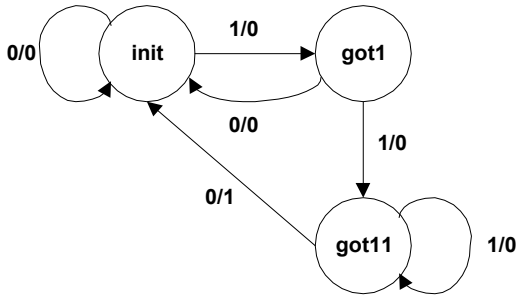
مثال ٢ : یک D-FF حساس به لبه‌ی بالارونده با reset آسنکرون طراحی کنید.

```
// Behavioral model of a d-ff with asynchronous reset

module d_ff ( d, clk, rst, q );
    input d, clk, rst;
    output q;
    reg q;

    always @( posedge clk or posedge rst)
    begin
       if (rst)
           q = 1'b0;
       else
           q = d;
    end

endmodule
```

مثال ۳ : یک شمارنده‌ی ٤ بیتی با پایه‌های سنکرون u_d ,ld و پایه‌ی rst آسنکرون طراحی کنید که با لبه‌ی پایین‌رونده‌ی clk کار کند.

```
// Behavioral model of a up/down counter

module counter ( clk, ld, rst, u_d, d_in, q );
    input clk, ld, rst, u_d;
    input [3:0] d_in;
    output [3:0] q;
    reg [3:0] q;

    always @( negedge clk or posedge rst)
    begin
       if (rst)
           q = 4'b0000;
       else if( ld )
           q = d_in;
       else if( u_d )
           q = q + 1;
       else
           q = q - 1;
    end

endmodule
```

مثال ٤ (طراحی دیاگرام حالت mealy): دیاگرام حالت زیر را مدلسازی کنید.

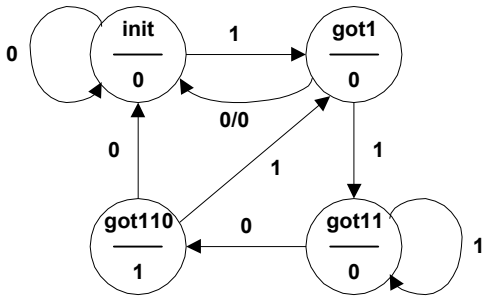


```
// Mealy state machine

`define init         2'd0
`define got1         2'd1
`define got11        2'd2

module seq_detector ( clk, x, rst, y );
    input clk, x, rst;
    output y;

    reg [1:0] cur_state;

    always @( posedge clk )
    begin
       if ( rst )
          cur_state = `init;
       else case ( cur_state )
          `init  : cur_state = x ? `got1  : `init;
          `got1  : cur_state = x ? `got11 : `init;
          `got11 : cur_state = x ? `got11 : `init;
       endcase
    end

    assign y = (cur_state==`got11 && x==1'b0) ? 1'b1 : 1'b0;

endmodule
```

مثال ۵ : دیاگرام حالت مثـال ٤ را بـا اسـتفاده از مـدل هـافمن مدلسـازی کنیـد.

```
// Mealy state machine (Hufmann Model)

`define init         2'd0
`define got1         2'd1
`define got11        2'd2

module seq_detector ( clk, x, rst, y );
    input clk, x, rst;
    output y;

    reg [1:0] present_state, next_state;

    always @( posedge clk )
        if(rst)
           present_state = `init;
        else
           present_state = next_state;

    always @( present_state or x )
    begin
        case ( present_state )
          `init  : next_state = x ? `got1  : `init;
          `got1  : next_state = x ? `got11 : `init;
          `got11 : next_state = x ? `got11 : `init;
       endcase
    end

    assign y = (present_state==`got11 && x==1'b0) ? 1'b1 : 1'b0;

endmodule
```

مثال ٦ (طراحی دیاگرام حالت moore): دیاگرام حالت زیر را مدلسازی کنید.




```
// Moore state machine

`define init         2'd0
`define got1         2'd1
`define got11        2'd2
`define got110       2'd3

module seq_detector ( clk, x, rst, y );
    input clk, x, rst;
    output y;

    reg [1:0] cur_state;

    always @( posedge clk )
    begin
       if ( rst )
          cur_state = `init;
       else case ( cur_state )
          `init   : cur_state = x ? `got1  : `init;
          `got1   : cur_state = x ? `got11 : `init;
          `got11  : cur_state = x ? `got11 : `got110;
          `got110 : cur_state = x ? `got1  : `init;
       endcase
    end

    assign y = (cur_state==`got110) ? 1'b1 : 1'b0;

endmodule
```

مثال ٧ : دیاگرام حالت مثال ٦ را با استفاده از مدل هافمن مدلسازی کنید.

```
// Moore state machine (Hufmann Model)

`define init         2'd0
`define got1         2'd1
`define got11        2'd2
`define got110       2'd3

module seq_detector ( clk, x, rst, y );
    input clk, x, rst;
    output y;

    reg [1:0] present_state, next_state;

    always @( posedge clk )
        if(rst)
           present_state = `init;
        else
           present_state = next_state;
```

```
        always @( present_state or x )
        begin
            case ( present_state )
              `init   : next_state = x ? `got1  : `init;
              `got1   : next_state = x ? `got11 : `init;
              `got11  : next_state = x ? `got11 : `got110;
              `got110 : next_state = x ? `got1  : `init;
           endcase
        end

        assign y = (present_state==`got110) ? 1'b1 : 1'b0;

    endmodule
```

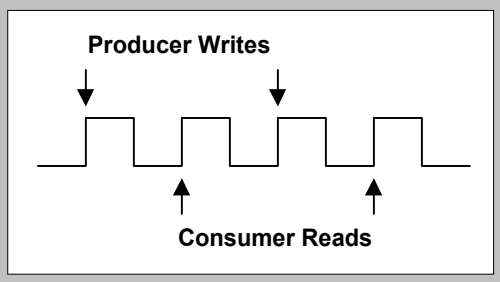
مثال۸: فرض کنید در یک سیستمی یک پروسس تولید کننده و یک پروسس مصرف کننده وجود دارد. می‌خواهیم Handshake بین این دو پروسس را به نوعی مدلسازی کنیم. سیستم به اینصورت عمل می‌کند که ابتدا پروسس تولیدکننده داده‌ای را تولید می‌کند، پروسس مصرف کننده این داده را مصرف می‌کند. مادامیکه پروسس مصرف کننده داده را مصرف نکرده، پروسس تولیدکننده دادهٔ بعدی را تولید نمی‌کند.

```
module producer( dataOut, prodReady, consReady )
   output [7:0] dataOut;
   output prodReady;
   reg [7:0] dataOut;
   reg prodReady;
   input consReady;

always
   begin
      prodReady = 0;        // Indicate nothing to transfer
      forever
      begin
         // ... Prduce data and put into temp
         // Wait for consumer ready
         wait( consReady ) dataOut = $random;


         // Indicate ready to transfer
         prodReady = 1;
         // Finish handshake
         wait( !consReady ) prodReady = 0;
      end
   end

endmodule

module consumer( dataIn, prodReady, consReady )
   input [7:0] dataIn;
   input prodReady;
   output consReady;
   reg consReady;

   reg [7:0] dataInCopy;
```

```
    always
    begin
       consReady = 1;         // Indicate nothing to transfer
       forever
       begin
          wait( prodReady ) dataInCopy = dataIn;
          // Indicate value consumed
          consReady = 0;
          // ... munch on data
          // Complete handshake
          wait( !prodReady ) consReady = 1;
       end
    end

endmodule

// Top-level Producer-Consumer process
module ProducerConsumer;
   wire [7:0] data;
   wire pReady, cReady;

   producer p( data, pReady, cReady );
   consumer c( data, pReady, cReady );

endmodule
```

پرسش ۶

- مثال تولیدکننده و مصرف‌کنندهٔ فوق را به نحوی تغییردهید که در لبهٔ بالارونده clk های متوالی تولیدکننده و مصرف‌کننده به ترتیب داده را تولید و مصرف کنند (شکل زیر را ببینید).

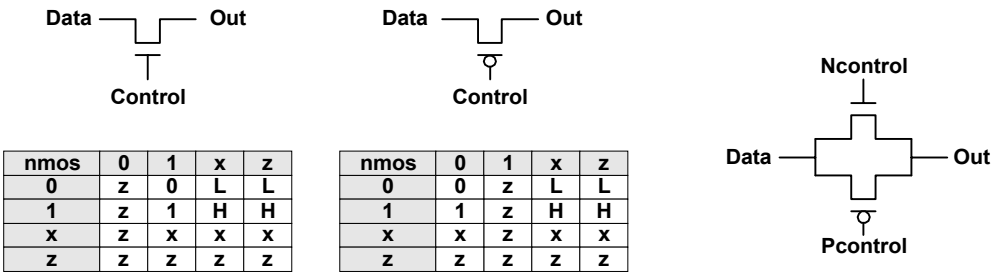


۸ مدلسازی در سطح سوئیچ

در Verilog برای انجام مدلسازی در سطح سوئیچ ساختارهایی از پیش تعبیه شده است، که در این قسمت به آنها می‌پردازیم.

۸-۱ سوئیچهای MOS, CMOS

در شکل ۱-۱۱ این سوئیچها و جدول صحت آنها نمایش داده شده است.



nmos	0	1	x	z
0	z	0	L	L
1	z	1	H	H
x	z	x	x	x
z	z	z	z	z

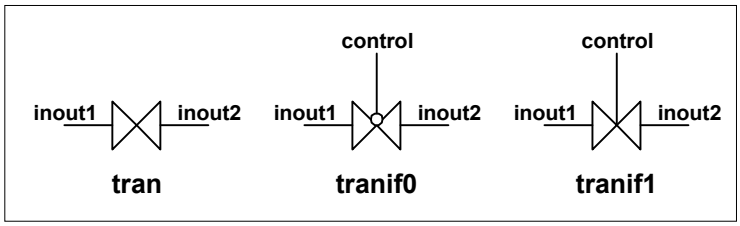
nmos	0	1	x	z
0	0	z	L	L
1	1	z	H	H
x	x	z	x	x
z	z	z	z	z

شکل ۱-۱۱- سوئیچهای MOS و جدول صحت آنها

```
nmos n1 ( out, data, control );
pmos ( out, data, control );
cmos c1 ( out, data, ncontrol, pcontrol );
```

۸-۲ سوئیچهای دوسویه

در شکل ۱-۱۲ سوئیچهای دوسویه نمایش داده شده است.



شکل ۱-۱۱- سوئیچهای MOS و جدول صحت آنها

```
tran t1 ( inout1, inout2 );
tranif0 ( inout1, inout2, control );
tranif1 ( inout1, inout2, control );
```

۸-۳ تغذیه و زمین

به هنگام استفاده از سوئیچها به منبع تغذیه و زمین (Vss, Vdd) نیاز داریم. برای این منظور دو کلمهٔ کلیدی supply0, supply1 به ترتیب برای Vss, Vdd درنظر گرفته شده است.

```
supply1 Vdd;
supply0 Vss;
assign a = Vdd;       // Connect a to Vdd
```

۸-٤ سوئیچهای مقاومتی

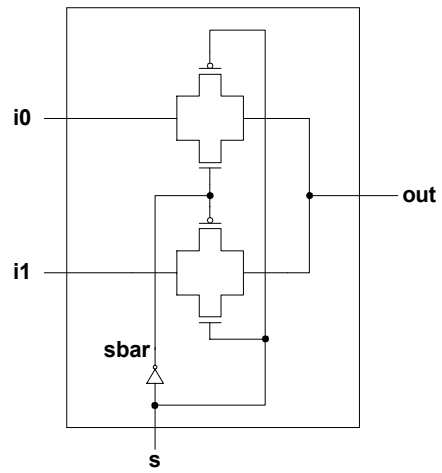
درحالات قبل سوئیچها ایده‌آل فرض شده بودند، یعنی هیچگونه تضعیفی در سیگنال عبوری از سوئیچ مشاهده نمی‌شد. ولی در سوئیچهای مقاومتی قدرت سیگنال عبوری از سوئیچ مطابق جدول ۱-۷ تغییر می‌کند. سوئیچهای مقاومتی با افزودن یک r به ابتدای نام سوئیچهای معمولی بدست می‌آیند.

```
rnmos, rpmos                 // Resistive nmos,pmos switches
rcmos                        // Resistive cmos switch
rtran, rtranif0, rtranif1    // Resistive tran,tranif0,tranif1 switches
```

Input Strength	Output Strength
supply	pull
strong	pull
pull	weak
weak	medium
large	medium
medium	small
small	small
high	high

جدول ۱-۷- جدول تضعیف سیگنال سوئیچهای مقاومتی

مثال : یک مالتی پلکسر ۲ به ۱ طراحی کنید.




```
module mux2_to_1 ( out, i0, i1, s );
   output out;
   input s, io, i1;

   wire sbar;

   not ( sbar, s );
   cmos ( out, i0, sbar, s );
   cmos ( out, i1, s, sbar );

endmodule
```