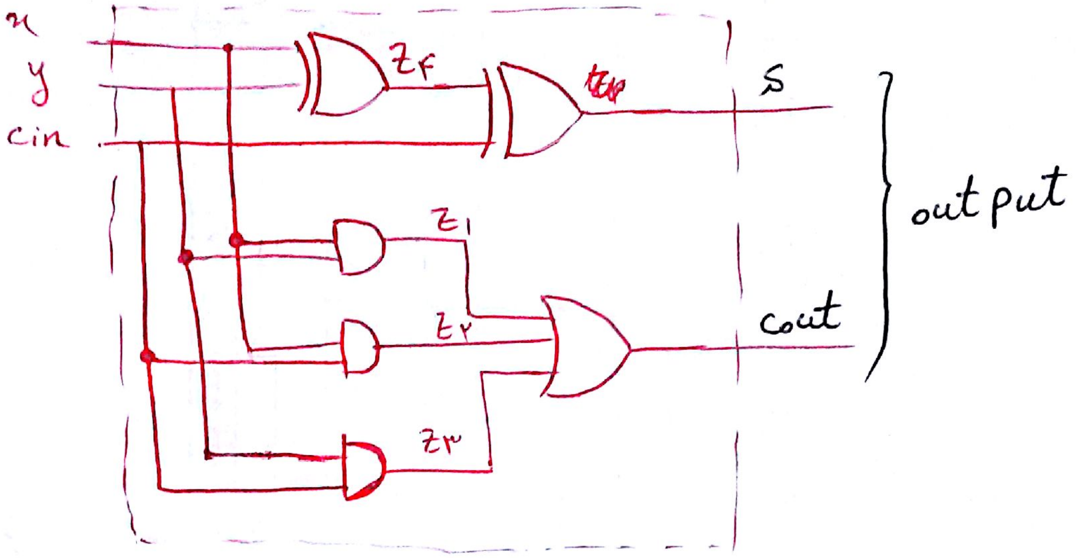
IC Dr. Shaabany Se 6 TA verilog




```
Module fulladd (cin, x, y, s, cout);
input cin, x, y;
output s, count;
assign s = x ^ y ^ cin ;
assign cout = (x & y) | (x & cin) | (y & cin);
end module
```

default 1 bit

* مهم : xor معادل جمع معمولی است که تنها بیت آخر آن را می‌دهد.

$1 \wedge 1 = 1 + 1 = 1\textcircled{0}$ ← بیت آخر

در اینجا cout رقم دوم ⟸ یک بیت دارد. ⟸ S که در بالا ⟸ S = x+y+cin ; ⟹

جمع x+y+cin از آن

می‌دهد یعنی مثلاً اگر هر سه ۱ باشند s=۱ و cout = ۱ (خاصیت این مدار full adder)

← پس معادل دو خط assign در برنامه می‌توان نوشت:

```
assign {cout, s} = x+y+cin;
```

Syntax:

تعریف gate

```
and And1 (z1, x, y)
```

```
Module fulladd (cin, x, y, s, cout)
input cin, x, y;
output s, cout;
wire z1, z2, z3, z4;
and And1 (z1, x, y);
and And2 (z2, x, cin);
and And3 (z3, y, cin);
or Or1 (cout, z1, z2, z3);
xor Xor1 (z4, x, y);
xor Xor2 (s, z4, cin);
endmodule
```

```
always @(xor y)
begin
s = x^y;
c = x&y;
end
```

(sensitivity_list) یعنی با تغییر عبارت داخل پرانتز عبارات داخل always هم تغییر می‌کند. مثلاً اینجا با تغییر x و y، s هم تغییر کند.

برای بیش از یک gate

```
if(expression 1)
  begin
  - - - -
  end
else if (expression 2)
  begin
  - ——— -
  end
 else
 begin
 ——— -
  end
```

```
always @ (S)
if (S==0)
   f=w0;
else
 f=w1;
```

always @* → حساس به همه‌ی متغیرهای داخل always

* دقت شود که ترتیب کدنویسی بسیار مهم است

* اگر else تعریف نشود به صورت پیش‌فرض خروجی به ورودی متصل می‌شود.

```
case (expression)
  alternative 1: begin
                 - - ——
                 end
  alternative 2: begin
                 ———
                 end
  default :  begin
             ———   → (اختیاری)
             end
  end case
```