Daniel Kostecki

Digital Systems Design Project 1

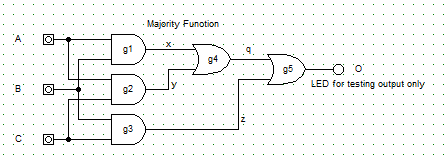
10-27-16

**3-Input Majority Function**

Our majority function will take 3 inputs and will return 1 or 0 based on which input there is the most of. For example, if there are two 1s passed and one 0 then the function will return 1, and if two 0s are passed and one 1 then the function will return 0. This makes the name of the function descriptive, as the name of the function describes that it returns the value for which a majority are input. This function was implemented with three AND gates and two OR gates.

|  |  |  |  |
| --- | --- | --- | --- |
| **3-Input Majority Function Truth Table** | | | |
| **A** | **B** | **C** | **O** |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **3-Input Majority Function Map** | | | | | |
| **A** | **BC** | | | | |
|  | **00** | **01** | **11** | **10** |
| **0** |  |  | 1 |  |
| **1** |  | 1 | 1 | 1 |



module majority(A,B,C,O);

input A,B,C;

output O;

wire x,y,z,q;

and g1 (x,A,B),

g2 (y,A,C),

g3 (z,B,C);

or g4 (q,x,y),

g5 (O,q,z);

endmodule

module test;

reg A,B,C; // Reg for inputs

wire O; // Wire for outputs

majority M(A,B,C,O);

initial

begin

$display("Time A B C O");

A=0; B=0; C=0;

#10 A=0; B=0; C=1;

#10 A=0; B=1; C=0;

#10 A=0; B=1; C=1;

#10 A=1; B=0; C=0;

#10 A=1; B=0; C=1;

#10 A=1; B=1; C=0;

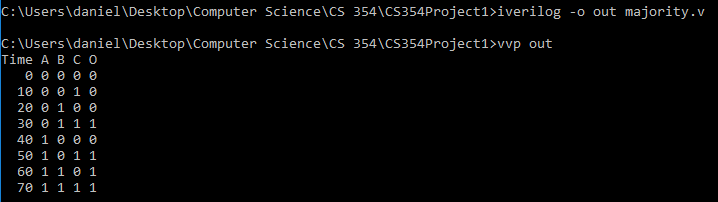
#10 A=1; B=1; C=1;

end

initial

$monitor("%4d %b %b %b %b",$time,A,B,C,O);

endmodule

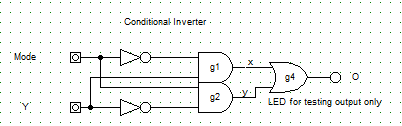


**Conditional Inverter (without XOR)**

The point of a conditional inverter is to have a mode input and a value input where when the mode is turned off the output of the function is just the value of the input. When the mode is one then the output is the opposite of the input. For example, if mode is 0 and Y is one then the function will return 1. When mode is 1 and Y is 1 then the function will return 0.

|  |  |  |
| --- | --- | --- |
| **Conditional Inverter Truth Table** | | |
| **Mode** | **Y** | **O** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Conditional Inverter Map** | | | |
| **Mode** | **Y** | | |
|  | **0** | **1** |
| **0** |  | 1 |
| **1** | 1 |  |



module conditional\_inverter(Mode,Y,O);

input Mode,Y;

output O;

wire x,y;

and g1 (x,!Mode,Y),

g2 (y,Mode,!Y);

or g4 (O,x,y);

endmodule

module test;

reg M,Y; // Reg for inputs

wire O; // Wire for outputs

conditional\_inverter C(M,Y,O);

initial

begin

$display("Time M Y O");

M=0; Y=0;

#10 M=0; Y=1;

#10 M=1; Y=0;

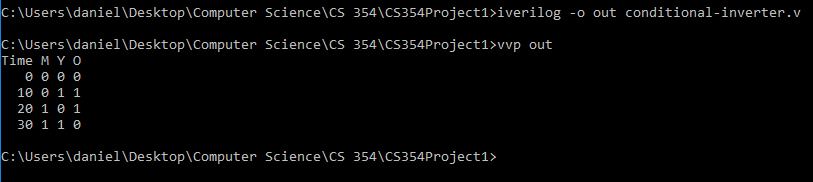
#10 M=1; Y=1;

end

initial

$monitor("%4d %b %b %b",$time,M,Y,O);

endmodule

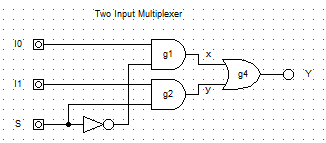


**Two-Input Multiplexer**

The Two-Input Multiplexer is a function that takes in two inputs and a variable that can be flipped using a third input. So, when S is off then you will only receive a 1 output for I0 or I0 with I1, not for I1 alone. When S is 1, then the opposite is true, and when I1 is 1 or I1 with I0 the function will return 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Two-Input Multiplexer Truth Table** | | | |
| **S** | **I0** | **I1** | **Y** |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Two-Input Multiplexer Map** | | | | | |
| **S** | **I0I1** | | | | |
|  | **00** | **01** | **11** | **10** |
| **0** |  |  | 1 | 1 |
| **1** |  | 1 | 1 |  |



module two\_input\_mux(S,I0,I1,Y);

input S,I0,I1;

output Y;

wire x,y;

and g1 (x,I0,!S),

g2 (y,I1,S);

or g4 (Y,x,y);

endmodule

module test;

reg s,i0,i1; // Reg for inputs

wire y; // Wire for outputs

two\_input\_mux T(s,i0,i1,y);

initial

begin

$display("Time S I0 I1 Y");

s=0; i0=0; i1=0;

#10 s=0; i0=0; i1=1;

#10 s=0; i0=1; i1=0;

#10 s=0; i0=1; i1=1;

#10 s=1; i0=0; i1=0;

#10 s=1; i0=0; i1=1;

#10 s=1; i0=1; i1=0;

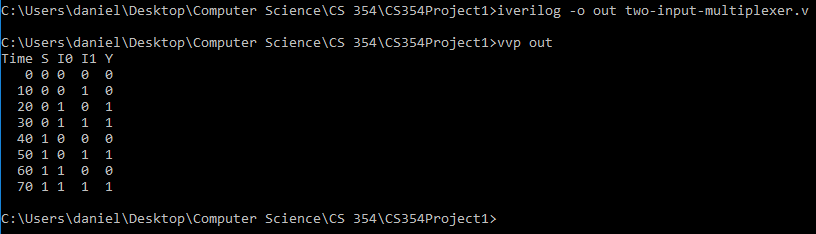
#10 s=1; i0=1; i1=1;

end

initial

$monitor("%4d %b %b %b %b",$time,s,i0,i1,y);

endmodule



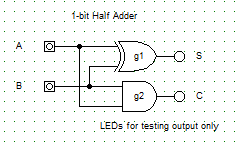
**1-Bit Half Adder**

A half adder takes in two inputs and has two outputs. The two inputs are the two values you would like to add. The two outputs are the sum and carry of the addition. For example, if A is 1 and B is 1 the S will be 0 and C will be 1. If A or B is 1 and the other is 0 then the S will be 1 and the C will be 0. If A and B are both 0 then S and C will be 0. In our case, we use the half adder as a constituent component of a full adder.

|  |  |  |  |
| --- | --- | --- | --- |
| **3-Input Majority Function Truth Table** | | | |
| **A** | **B** | **S** | **C** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| **1-Bit Half Adder S** | | | |
| **A** | **B** | | |
|  | **0** | **1** |
| **0** |  | 1 |
| **1** | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **1-Bit Half Adder C** | | | |
| **A** | **B** | | |
|  | **0** | **1** |
| **0** |  |  |
| **1** |  | 1 |



module half\_adder(A,B,S,C);

input A,B;

output S,C;

xor g1 (S,A,B);

and g2 (C,A,B);

endmodule

module test;

reg A,B; // Reg for inputs

wire S,C; // Wire for outputs

half\_adder M(A,B,S,C);

initial

begin

$display("Time A B S C");

A=0; B=0;

#10 A=0; B=1;

#10 A=1; B=0;

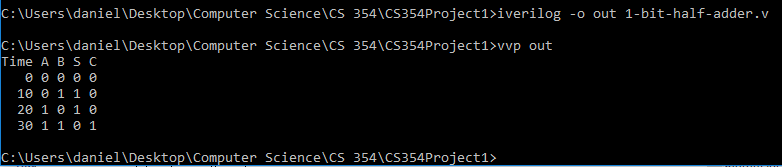
#10 A=1; B=1;

end

initial

$monitor("%4d %b %b %b %b",$time,A,B,S,C);

endmodule



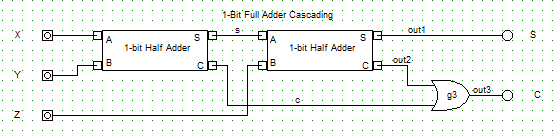
**1-Bit Full Adder Cascading**

A cascading 1-Bit Full Adder uses two half adders together to perform addition on three inputs are results in two outputs. This accounts for both the sum and the carry of the full adder. Because in this example we are using two half adders, we use module for this and then reference it in the adder.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1-Bit Full Adder Cascading Truth Table** | | | | |
| **X** | **Y** | **Z** | **S** | **C** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1-Bit Full Adder Cascading Map S** | | | | | |
| **X** | **YZ** | | | | |
|  | **00** | **01** | **11** | **10** |
| **0** |  | 1 |  | 1 |
| **1** | 1 |  | 1 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1-Bit Full Adder Cascading Map C** | | | | | |
| **X** | **YZ** | | | | |
|  | **00** | **01** | **11** | **10** |
| **0** |  |  | 1 |  |
| **1** |  | 1 | 1 | 1 |



module half\_adder(A,B,S,C);

input A,B;

output S,C;

xor g1 (S,A,B);

and g2 (C,A,B);

endmodule

module add(X,Y,Z,out1,out3);

input X,Y,Z;

output out1,out2,out3;

wire s,c;

half\_adder half1 (X,Y,s,c),

half2 (s,Z,out1,out2);

or g3(out3,out2,c);

endmodule

module test;

reg X,Y,Z; // Reg for inputs

wire S,C; // Wire for outputs

add M(X,Y,Z,S,C);

initial

begin

$display("Time X Y Z S C");

X=0; Y=0; Z=0;

#10 X=0; Y=0; Z=1;

#10 X=0; Y=1; Z=0;

#10 X=0; Y=1; Z=1;

#10 X=1; Y=0; Z=0;

#10 X=1; Y=0; Z=1;

#10 X=1; Y=1; Z=0;

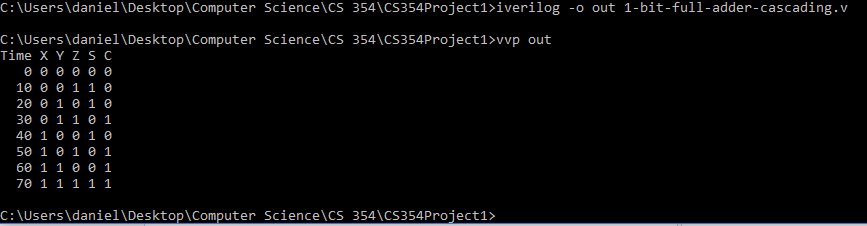
#10 X=1; Y=1; Z=1;

end

initial

$monitor("%4d %b %b %b %b %b",$time,X,Y,Z,S,C);

endmodule

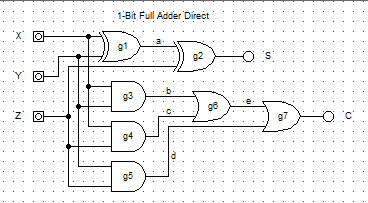


**1-Bit Full Adder Cascading**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1-Bit Full Adder Cascading Truth Table** | | | | |
| **X** | **Y** | **Z** | **S** | **C** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1-Bit Full Adder Cascading Map S** | | | | | |
| **X** | **YZ** | | | | |
|  | **00** | **01** | **11** | **10** |
| **0** |  | 1 |  | 1 |
| **1** | 1 |  | 1 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1-Bit Full Adder Cascading Map C** | | | | | |
| **X** | **YZ** | | | | |
|  | **00** | **01** | **11** | **10** |
| **0** |  |  | 1 |  |
| **1** |  | 1 | 1 | 1 |



module add(X,Y,Z,S,C);

input X,Y,Z;

output S,C;

wire a,b,c,d,e;

xor g1(a,X,Y),

g2(S,a,Z);

and g3(b,X,Y),

g4(c,X,Z),

g5(d,Y,Z);

or g6(e,b,c),

g7(C,e,d);

endmodule

module test;

reg X,Y,Z; // Reg for inputs

wire S,C; // Wire for outputs

add M(X,Y,Z,S,C);

initial

begin

$display("Time X Y Z S C");

X=0; Y=0; Z=0;

#10 X=0; Y=0; Z=1;

#10 X=0; Y=1; Z=0;

#10 X=0; Y=1; Z=1;

#10 X=1; Y=0; Z=0;

#10 X=1; Y=0; Z=1;

#10 X=1; Y=1; Z=0;

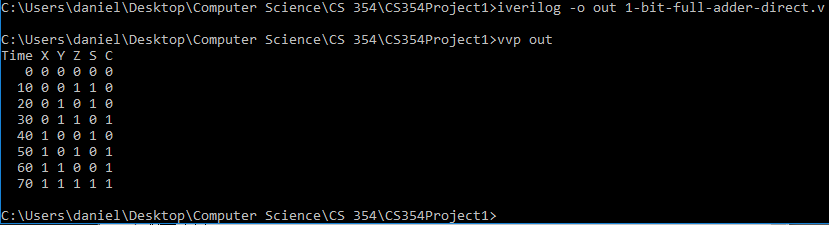
#10 X=1; Y=1; Z=1;

end

initial

$monitor("%4d %b %b %b %b %b",$time,X,Y,Z,S,C);

endmodule



**4-Bit Adder/Subtractor**