3주차 결과보고서

학번 : 20211531

이름 : 나호영

1.

코드를 작성하고 시뮬레이션을 마친 후, FPGA에서 직접 작동시키려면 Synthesis(합성) 과정이 필요하다. 파일을 저장한 후 Run Synthesis 버튼을 누르면 합성이 완료된다. 그 후에 I/O input을 설정해준다. PinList를 참고하여 원하는 input과 output을 할당한 후 저장한다. 이후 구현(Implementation)하기 위해 Run Implementation 버튼을 누르면 이전에 완료했던 핀 할당으로 인해 재합성되고 구현이 된다. 구현이 완료되면 비트스트림(Bitstream)을 생성해야 한다. Generate Bitstream 버튼을 누르면 비트스트림이 생성된다. 마지막으로 Target 연결, 즉 우리가 사용할 FPGA와 연결해야하는데 Open target -> Auto Connect 버튼을 누르면 프로그램이 자동으로 컴퓨터와 연결한 FPGA 보드를 찾는다. 그 후 Program Device 버튼을 눌러 비트스트림 파일과 디버그 파일 경로를 설정하고 Program 버튼을 누르면 FPGA를 작동시킬 수 있다.

2.

아래는 3-input AND gate의 모듈 코드이다. assgin을 이용한 연속 할당문으로 output을 정의하였다.

`timescale 1ns / 1ps

module inv(

input A,

input B,

input C,

output D,

output E

);

assign D = A&&B;

assign E = C&&D;

endmodule

이를 시뮬레이션하기 위하여 작성한 테이블벤치는 아래와 같다. input인 A,B,C에 1을 할당하고 각각 20, 30, 40, 50의 time마다 not값을 할당하게 하였다.

`timescale 1ns / 1ps

module inv\_tb;

reg A;

reg B;

reg C;

wire D;

wire E;

inv u\_inv(

.A ( A ),

.B ( B ),

.C ( C ),

.D ( D ),

.E ( E )

);

initial begin

A=1'b1;

B=1'b1;

C=1'b1;

end

always begin

A=#20 ~A;

B=#30 ~B;

C=#40 ~C;

end

initial begin

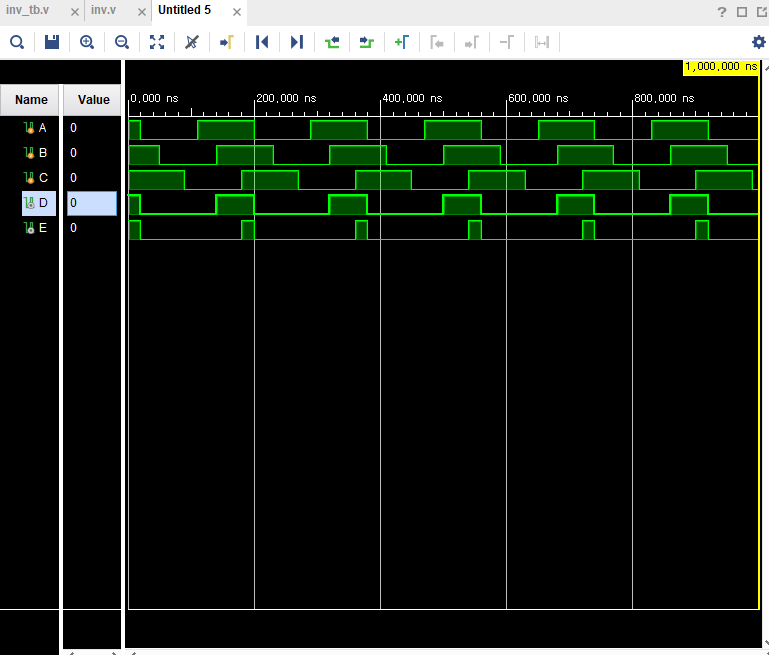
#1000

$finish;

end

endmodule

시뮬레이션 결과는 아래와 같다.



결과적으로 D는 A와 B가 high일 때, E는 D와 C가 high일 때 high가 된다. 결과를 바탕으로 작성한 진리표는 아래와 같다.

| In A | In B | In C | Out D | Out E |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

3.

아래는 4-input AND gate의 모듈 코드이다. assgin을 이용한 연속 할당문으로 output을 정의하였다.

`timescale 1ns / 1ps

module inv(

input A,

input B,

input C,

input D,

output E,

output F,

output G

);

assign E = A&&B;

assign F = E&&C;

assign G = F&&D;

endmodule

이를 시뮬레이션하기 위하여 작성한 테이블벤치는 아래와 같다. input인 A,B,C,D에 1을 할당하고 각각 20, 30, 40, 50의 time마다 not값을 할당하게 하였다.

`timescale 1ns / 1ps

module inv\_tb;

reg A;

reg B;

reg C;

reg D;

wire E;

wire F;

wire G;

inv u\_inv(

.A ( A ),

.B ( B ),

.C ( C ),

.D ( D ),

.E ( E ),

.F ( F ),

.G ( G )

);

initial begin

A=1'b1;

B=1'b1;

C=1'b1;

D=1'b1;

end

always begin

A=#20 ~A;

B=#30 ~B;

C=#40 ~C;

D=#50 ~D;

end

initial begin

#1000

$finish;

end

endmodule

시뮬레이션 결과는 아래와 같다.



결과적으로 E는 A와 B가 high일 때, F는 C와 E가 high일 때, G는 F와 D가 high일 때 high가 된다. 결과를 바탕으로 작성한 진리표는 아래와 같다.

| In A | In B | In C | In D | Out E | Out F | Out G |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

4.

아래는 3-input OR gate의 모듈 코드이다. assgin을 이용한 연속 할당문으로 output을 정의하였다.

`timescale 1ns / 1ps

module inv(

input A,

input B,

input C,

output D,

output E

);

assign D = A||B;

assign E = C||D;

endmodule

이를 시뮬레이션하기 위하여 작성한 테이블벤치는 아래와 같다. input인 A,B,C에 1을 할당하고 각각 20, 30, 40, 50의 time마다 not값을 할당하게 하였다.

`timescale 1ns / 1ps

module inv\_tb;

reg A;

reg B;

reg C;

wire D;

wire E;

inv u\_inv(

.A ( A ),

.B ( B ),

.C ( C ),

.D ( D ),

.E ( E )

);

initial begin

A=1'b1;

B=1'b1;

C=1'b1;

end

always begin

A=#20 ~A;

B=#30 ~B;

C=#40 ~C;

end

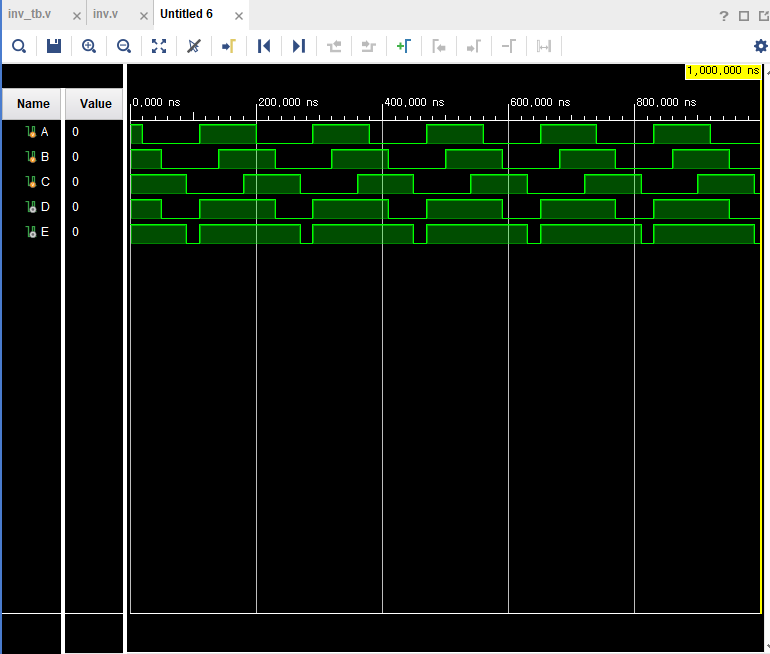
initial begin

#1000

$finish;

end

시뮬레이션 결과는 아래와 같다.



결과적으로 D는 A와 B 둘중 하나라도 high일 때, E는 D와 C 둘중 하나라도 high일 때 high가 된다. 결과를 바탕으로 작성한 진리표는 아래와 같다.

| In A | In B | In C | Out D | Out E |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

5.

아래는 4-input OR gate의 모듈 코드이다. assgin을 이용한 연속 할당문으로 output을 정의하였다.

`timescale 1ns / 1ps

module inv(

input A,

input B,

input C,

input D,

output E,

output F,

output G

);

assign E = A||B;

assign F = E||C;

assign G = F||D;

endmodule

이를 시뮬레이션하기 위하여 작성한 테이블벤치는 아래와 같다. input인 A,B,C,D에 1을 할당하고 각각 20, 30, 40, 50의 time마다 not값을 할당하게 하였다.

`timescale 1ns / 1ps

module inv\_tb;

reg A;

reg B;

reg C;

reg D;

wire E;

wire F;

wire G;

inv u\_inv(

.A ( A ),

.B ( B ),

.C ( C ),

.D ( D ),

.E ( E ),

.F ( F ),

.G ( G )

);

initial begin

A=1'b1;

B=1'b1;

C=1'b1;

D=1'b1;

end

always begin

A=#20 ~A;

B=#30 ~B;

C=#40 ~C;

D=#50 ~D;

end

initial begin

#1000

$finish;

end

endmodule

시뮬레이션 결과는 아래와 같다.

결과적으로 E는 A와 B 둘 중 하나라도 high일 때, F는 C와 E 둘중 하나라도 high일 때, G는 F와 D 둘중 하나라도 high일 때 high가 된다. 결과를 바탕으로 작성한 진리표는 아래와 같다.

| In A | In B | In C | In D | Out E | Out F | Out G |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

6.

먼저 AND Gate부터 살펴보면, 3-input AND Gate는 하나의 AND 게이트에 모든 input 값을 넣는 A와 비교해보자. A는 input 값이 모두 1인 경우를 제외하고는 최종 결과값인 D가 0이다. 최종 결과값인 E도 마찬가지로 input값이 모두 1인 경우를 제외하고는 모두 0이므로 각각의 input 값에 따라 결과값이 똑같이 도출된다고 볼 수 있다. 4-input AND Gate도 input값이 모두 1일 때 결과값이 1로 도출되는데 마찬가지로 A의 진리표와 비교했을 때 각각의 input에 따른 최종 결과값이 같다. OR Gate를 살펴보자. 3-input OR Gate는 input값 3개중 하나라도 1이면 결과값이 1이다. A와 진리표와 비교했을 때, input에 따른 결과값이 같다. 4-input OR Gate는 4개 중 하나라도 1이면 결과값이 1이다. 역시 A의 진리표와 비교해보면 마찬가지로 input 중 하나라도 1이 있다면 최종 결과값인 G가 1로 도출되었다.

이번 실험에서는 하나의 실험에 한가지의 종류의 게이트를 이용한 실습이었다. 따라서 모든 input을 하나의 Gate에 넣어서 Output을 도출하는 설계말고 여러가지 Gate를 거치며 중간 Output을 도출하는 설계를 해도 결과값이 똑같이 나왔다. 하지만 만약 여러가지 종류의 Gate를 섞은 설계를 가정해보면 다양한 결과값들과 Logic이 나올 것을 예상할 수 있다.

7.

++다단 게이트 회로

다단 게이트 회로란 2단 이상의 회로를 말한다. 이번 실험에서는 같은 종류의 Gate로 구성했지만 AND와 OR 게이트를 섞어서 다단 게이트 회로를 구성할 수 있다. 이 때, 2단 이상의 회로의 종류는 총 3가지로, 그 종류는 AND-OR / OR-AND / OR-AND-OR 이다. 이를 살펴보면 여러가지 회로를 설계할 수 있다는 사실을 알 수 있다. 당연히 다음 실험에서 살펴볼 NAND 게이트와 NOR 게이트를 이용하여 역으로 AND, OR 게이트를 설계할 수 있다. 실제로 NAND와 NOR 게이트를 많이 이용하는데 그 이유는 앞서 말했던 것처럼 여러가지 Gate를 조합하여 어떤 논리도 구현할 수 있고 AND나 OR 게이트보다 훨씬 속도가 빠르며 더욱 적은 비용과 부품을 사용하기 때문이다.