3주차 결과보고서

전공: 신문방송학과 학년: 4학년 학번: 20191150 이름: 전현길

1. FPGA 동작법을 설명하시오.

FPGA의 동작 검증 과정은 강의자료에 따르면 **Verilog coding, Run synthesis, Device/Pin assignment, Synthesis/Implement, Device configuration** 과정으로 구분된다.

1) Verilog Coding

Vivado project를 생성한 뒤 하드웨어 기술 언어(이 경우 Verilog)를 바탕으로 FPGA 내부의 논리 회로를 설계한다. 이 과정에서 Verilog로 source code, testbench code를 각각 작성하게 된다.

2) Run Synthesis

logic-level에서 회로를 기술하는 Verilog 소스 코드를 FPGA에서 동작할 수 있는 gate-level 코드로 변환하는 과정이다. 소프트웨어 코드의 compile 과정과 유사하다. 합성의 결과로 gate-level의 네트리스트(netlist)가 생성된다.

3) Device/Pin assignment

설계한 논리적 회로를 물리적 회로로 바꾸기 위해서는, 어떤 FPGA를 고를지(device assignment)와 어떤 핀에 각 변수를 할당할지(pin assignment)를 결정해야 한다. 이 때 **핀(pin)**이란 **FPGA 칩과 외부 장치(센서, 메모리, 디스플레이 등)와의 입출력 연결점**을 의미한다. FPGA 칩(Verilog를 통해 설계한 회로)에 연결된 입력 핀의 값에 따라 출력 핀이 결정되면, 이 값에 따라서 센서, 메모리, 디스플레이 등이 제어된다.

device assignment는 처음 프로젝트를 생성할 때 target device를 결정하거나, project manager - settings - general로 이동해 Pinned device를 선택해 수행할 수 있다. 이번 수업 때는 주로 xc7a75tfgg484-1을 사용한다.

Pin assignment는 add source - add or create constraints - create file로 이동해 constraints 파일(.xdc 확장자)을 생성한 뒤, 다음과 같은 할당문을 사용해 Pin과 Verilog source code의 port를 연결할 수 있다.

```
set_property -dict {PACKAGE_PIN (핀명] IOSTANDARD LVCMOS33}
```

[get_ports (변수명)]

이후에 Window - I/O Ports로 이동하면 입출력 pin과 port가 제대로 연결되었는지 확인할 수 있다.

4) Synthesis/Implement

이 과정에서 화면 좌측의 Run Implementation, Generate Bitstream - Open Hardware Manager을 차례로 실행하여 Synthesis/Implement 과정을 수행할 수 있다. 이 작업은 gate-level에서 synthesis된 회로를 바탕으로 모듈과 전선을 배치(routing)하는 작업이다.

5) Device configuration

화면 좌측의 Run Synthesis, Run Implementation, Generate Bitstream - Open Hardware Manager을 차례로 실행하여 Synthesis/Implement 과정을 수행할 수 있다. 이 작업은 앞서 생성된 회로 및 모듈, 전선의 물리적 배치(routing) 정보를 담은 bitstream file을 FPGA에 다운로드하는 과정이다.

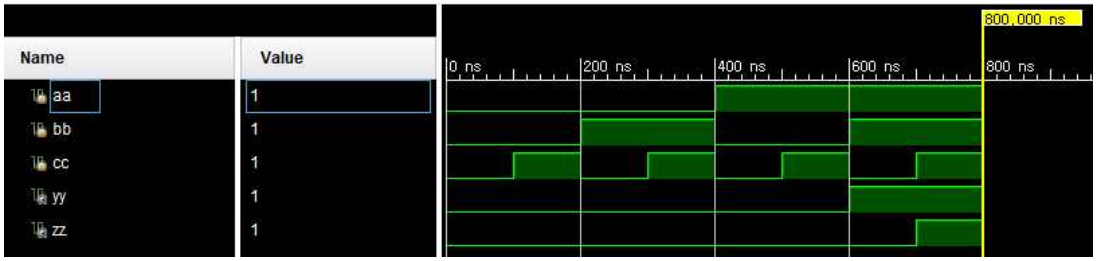
6) 실제 FPGA 동작 검증

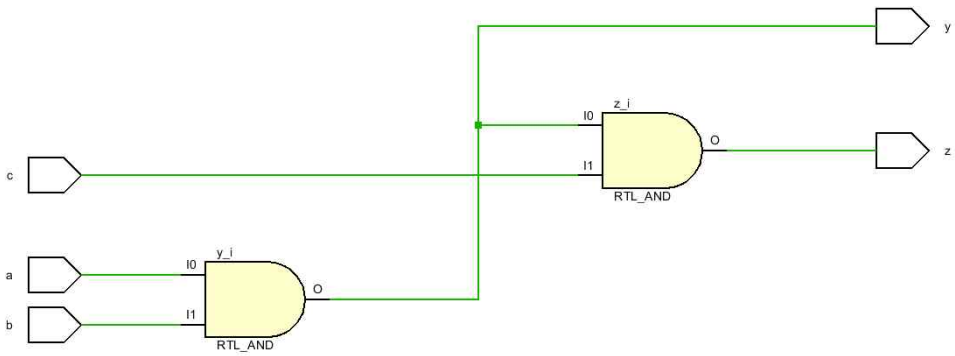
a) open target - auto connect를 클릭한다.

b) (project명) - (project명)_runs - impl_1 디렉토리에 debug_nets.ltx 파일을 넣어 준다.

c) program device로 이동하여 앞서 생성한 bitstream file(.bit)과 debug probes files(.ltx)을 지정해 준 뒤 Program 버튼을 눌러 예상한 대로 동작하는지 확인한다.

2. 3-input AND gate의 simulation 결과 및 과정에 대해서 설명하시오. (3 input-2 output, 진리표 작성)

3-Input AND gate simulation


3-Input AND gate circuit design


3-Input AND gate source	3-Input AND gate testbench source

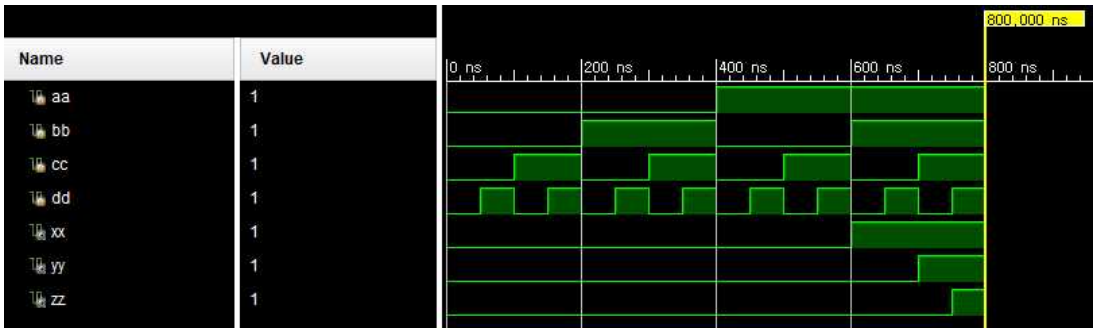
`timescale 1ns / 1ps
 	
module AND_three(
input a, b, c,
output y, z
);
 	
assign y = a&b;
assign z = y&c;
 	
endmodule
	`timescale 1ns / 1ps
	
	module AND_three_tb;
	
	reg aa, bb, cc;
	wire yy, zz;
	
	AND_three u_test(
	.a (aa), .b (bb), .c(cc),
	.y (yy), .z (zz)
	);
	
	initial begin
	aa = 1'b0; bb = 1'b0; cc = 1'b0;
	end
	
	always@(aa or bb or cc) begin
	aa <= #400 ~aa; bb <= #200 ~bb;
	cc <= #100 ~cc;
	end
	
	initial begin
	#800
	$finish;
	end
	
	endmodule

3-2 AND truth table				
input			output	
a	b	c	y	z
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

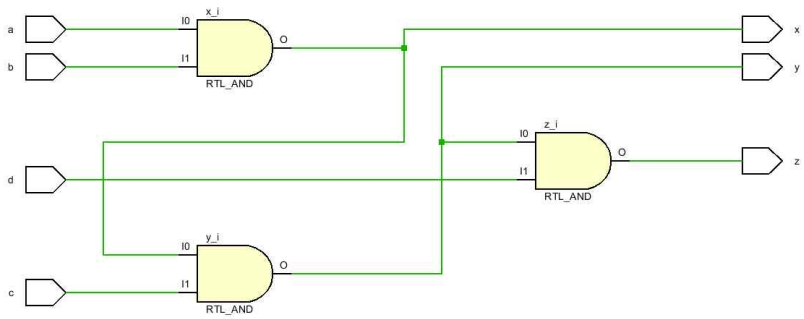
시뮬레이션의 편의성을 위해, 진리표와 동일한 형식대로 값이 변화하도록 테스트벤치 코드를 작성했다. y값은 a, b가 모두 1인 경우에만 1이 되며 그 외엔 0인 것을 확인할 수 있다. z값은 a, b, c가 모두 1인 경우에만 1이 되며 그 외에는 0이다.

3. 4-input AND gate의 simulation 결과 및 과정에 대해서 설명하시오. (4 input-3 output, 진리표 작성)

4-Input AND gate simulation



4-Input AND gate circuit design



4-Input AND gate source

```
`timescale 1ns / 1ps

module AND_four(
      input a, b, c, d,
      output x, y, z
      );

assign x = a&b;
assign y = x&c;
assign z = y&d;

endmodule
```

4-Input AND gate testbench source

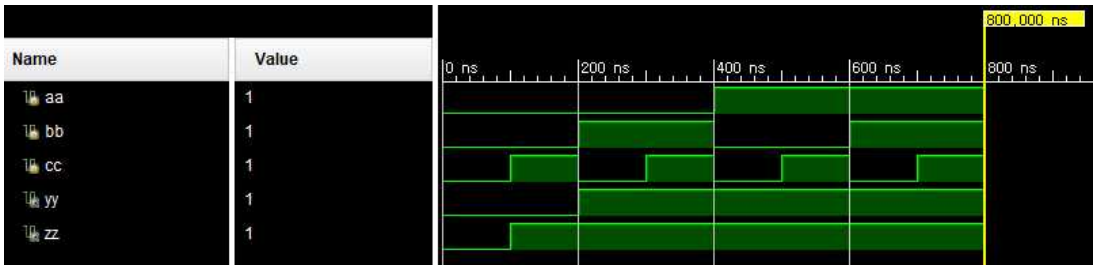
```
`timescale 1ns / 1ps

module AND_four_tb;

reg aa, bb, cc, dd;
wire xx, yy, zz;

AND_four u_test(
.a (aa), .b (bb), .c (cc), .d (dd),
.x (xx), .y (yy), .z (zz)
);

initial begin
      aa = 1'b0; bb = 1'b0; cc = 1'b0; dd = 1'b0;
end

always@(aa or bb or cc or dd) begin
      aa <= #400 ~aa; bb <= #200 ~bb;
      cc <= #100 ~cc; dd <= #50 ~dd;
end

initial begin
      #800
      $finish;
end

endmodule
```

4-3 AND truth table						
input				output		
a	b	c	d	x	y	z
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

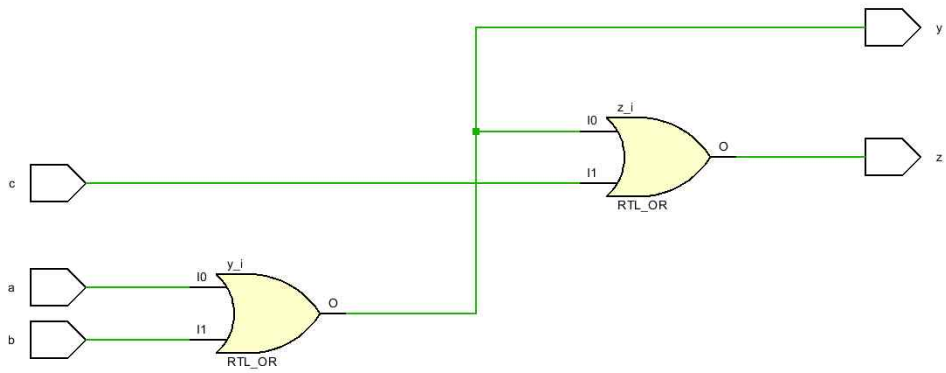
시뮬레이션의 편의성을 위해, 진리표와 동일한 형식대로 값이 변화하도록 테스트벤치 코드를 작성했다. x값은 a, b가 모두 1인 경우에만 1이 되며 그 외에는 0이다. y값은 a, b, c가 모두 1인 경우에만 1이 되며 그 외에는 0이다. z값은 a, b, c, d가 모두 1인 경우에만 1이 되며 그 외에는 0이다.

4. 3-input OR gate의 simulation 결과 및 과정에 대해서 설명하시오. (3 input-2 output, 진리표 작성)

3-Input OR gate simulation



3-Input OR gate circuit design



3-Input OR gate source	3-Input OR gate testbench source

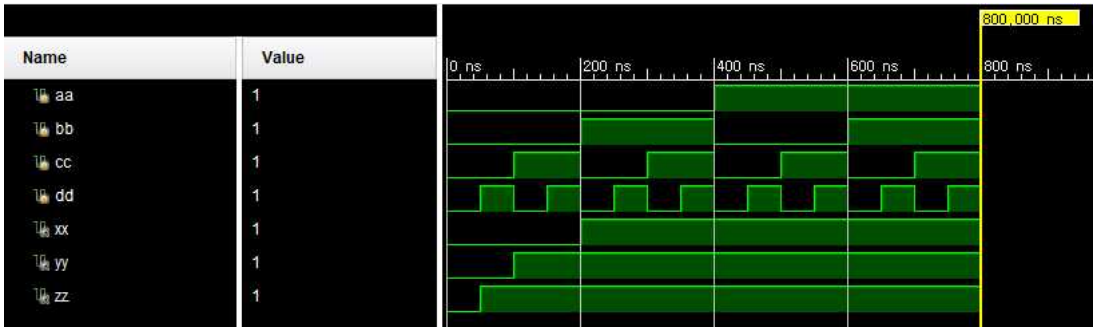
`timescale 1ns / 1ps
 	
module OR_three(
input a, b, c,
output y, z
);
 	
assign y = a|b;
assign z = y|c;
 	
endmodule
	`timescale 1ns / 1ps
	
	module OR_three_tb;
	
	reg aa, bb, cc;
	wire yy, zz;
	
	OR_three u_test(
	.a (aa), .b (bb), .c(cc),
	.y (yy), .z (zz)
	);
	
	initial begin
	aa = 1'b0; bb = 1'b0; cc = 1'b0;
	end
	
	always@(aa or bb or cc) begin
	aa <= #400 ~aa; bb <= #200 ~bb;
	cc <= #100 ~cc;
	end
	
	initial begin
	#800
	$finish;
	end
	
	endmodule

3-2 OR truth table				
input			output	
a	b	c	y	z
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

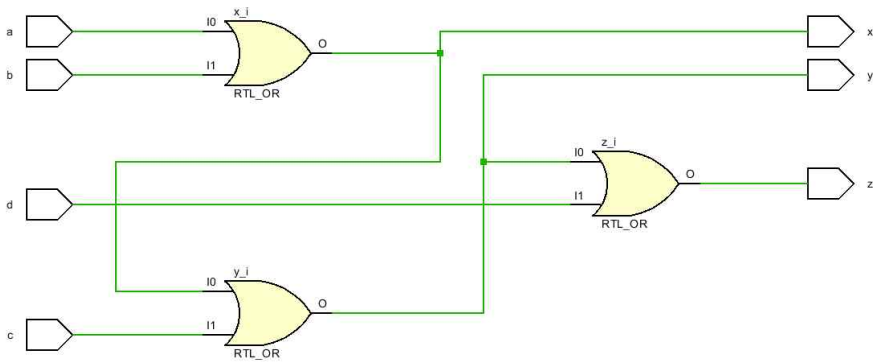
시뮬레이션의 편의성을 위해, 진리표와 동일한 형식대로 값이 변화하도록 테스트벤치 코드를 작성했다. x값은 a, b가 모두 0인 경우에만 0이 되며 그 외에는 1이다. y값은 a, b, c가 모두 0인 경우에만 0이 되며 그 외에는 1이다.

5. 4-input OR gate의 simulation 결과 및 과정에 대해서 설명하시오. (3 input-3 output, 진리표 작성)

4-Input OR gate simulation



4-Input OR gate circuit design



4-Input OR gate source

```
`timescale 1ns / 1ps

module OR_four(
    input a, b, c, d,
    output x, y, z
    );

assign x = a|b;
assign y = x|c;
assign z = y|d;

endmodule
```

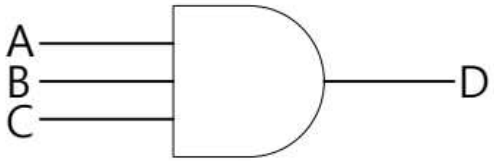
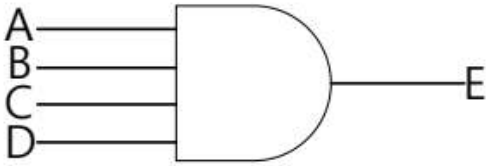
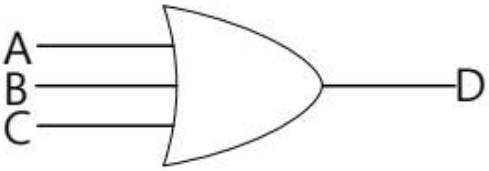
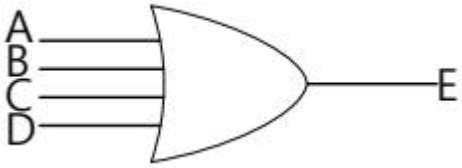
4-Input OR gate testbench source

```
`timescale 1ns / 1ps

module OR_four_tb;

reg aa, bb, cc, dd;
wire xx, yy, zz;

OR_four u_test(
.a (aa), .b (bb), .c (cc), .d (dd),
.x (xx), .y (yy), .z (zz)
);

initial begin
    aa = 1'b0; bb = 1'b0; cc = 1'b0; dd = 1'b0;
end

always@(aa or bb or cc or dd) begin
    aa <= #400 ~aa; bb <= #200 ~bb;
    cc <= #100 ~cc; dd <= #50 ~dd;
end

initial begin
    #800
    $finish;
end

endmodule
```

4-3 OR truth table						
input				output		
a	b	c	d	x	y	z
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

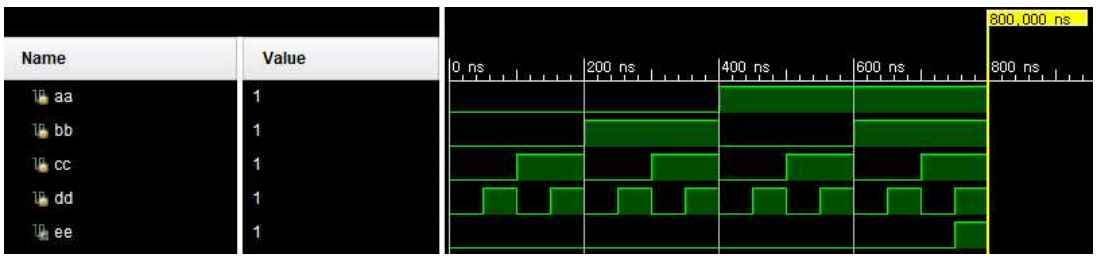
시뮬레이션의 편의성을 위해, 진리표와 동일한 형식대로 값이 변화하도록 테스트벤치 코드를 작성했다. x값은 a, b가 모두 0인 경우에만 0이 되며 그 외에는 1이다. y값은 a, b, c가 모두 0인 경우에만 0이 되며 그 외에는 1이다. z값은 a, b, c, d가 모두 0인 경우에만 0이 되며 그 외에는 1이다.

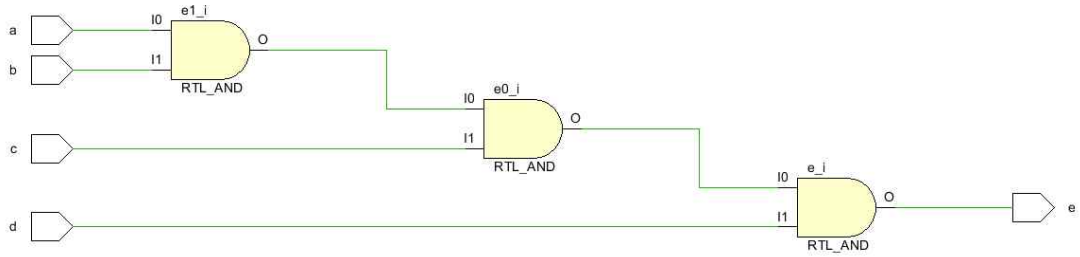
6. 결과 검토 및 논의 사항

3-Input AND	4-Input AND
	
3-Input OR	**4-Input OR**
	

3주차 실험에서는 3-Input/4-input AND/OR 게이트를 각각 Verilog로 구현한 뒤 시뮬레이션을 돌려 보고, 회로가 어떻게 설계되었는지, 진리표는 어떠한지 살펴보았다. 강의자료를 확인하면 한 logic gate에 입력값을 3개, 4개씩 주는 경우도 찾아볼 수 있는데, 이 방식으로도 회로를 설계하고 시뮬레이션을 돌려 보았다. 단, 모든 게이트를 설계하는 대신 4-input AND 게이트 하나만 설계해 보았다.

결과는 어렵지 않게 예상할 수 있는 것처럼, 우리가 미리 만들었던 4-input AND gate 회로도와 유사하게 구현되었다. 이는 실제 하드웨어에 4개의 입력을 직접 처리하는 게이트를 사용하지 않기 때문이다.

4-Input AND gate simulation


4-Input AND gate circuit design


4-Input OR gate source	4-Input OR gate testbench source

`timescale 1ns / 1ps
 	
module AND_four(
input a, b, c, d,
output e
);
 	
assign e = a&b&c&d;
 	
endmodule
	`timescale 1ns / 1ps
	
	module AND_four_tb;
	
	reg aa, bb, cc, dd;
	wire ee;
	
	AND_four u_test(
	.a (aa), .b (bb), .c (cc), .d (dd),
	.e (ee)
	);
	
	initial begin
	aa = 1'b0; bb = 1'b0; cc = 1'b0; dd = 1'b0;
	end
	
	always@(aa or bb or cc or dd) begin
	aa <= #400 ~aa; bb <= #200 ~bb;
	cc <= #100 ~cc; dd <= #50 ~dd;
	end
	
	initial begin
	#800
	$finish;
	end
	
	endmodule

7. 추가 이론 조사 및 작성

3, 4개의 입력을 직접 처리하는 회로를 FPGA상에서 구현하지 못하는 이유는, 전력 효율성 및 하드웨어 복잡도 문제 때문이다. 2개 이상의 입력 핀을 갖는 게이트를 구현하기 위해선 더욱 많은 트랜지스터를 필요로 하고, 이는 스위칭 전력 소비를 높이기 때문에 비효율적이다. 효율성을 제외하고도, 다중 입력 게이트를 사용하는 대신, 2-input 게이트들을 많이 사용하는 것이 설계될 수 있는 다양한 회로에 더욱 유연하게 대응할 수 있다.

이를 바탕으로 설령 Verilog 소스 코드에서 3-input, 4-input 논리 게이트를 설계했다 하더라도 FPGA 상에 실제 논리적 회로가 그대로 구현되는 것이 아니라는 점을 다시 한 번 확인할 수 있다. FPGA 상에서 구현되는 회로는 Synthesis 과정을 거쳐 생성된 gate-level 회로이다.