5주차 결과보고서

전공: 아트엔테크놀로지 학년: 3학년 학번: 20191172 이름: 함승우

1.

De Morgan 법칙, 제1법칙과 제2법칙을 Verilog 코드로 구현하여, 증명하고, 논리곱과 논리합 그리고 NOT 연산으로 이루어져 있는 수식을 De morgan 법칙을 적용해서 만든 또 다른 수식과의 비교를 통해서 De morgan 법칙을 학습한다. 1 – Bit 비교기를 코드로 구현하고, 회로의 동작을 살펴본다.

2.

(NAND,NOR 과 비교 포함)

텍스트, 도표, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

드 모르간의 제 1법칙은 ‘A+B의 보수 취한 것이 A의 보수와 B의 보수와 곱한 것과 같다’이고 제 2법칙은 ‘A\*B의 보수 취한 것이 A의 보수와 B의 보수와 합한 것과 같다’이다.

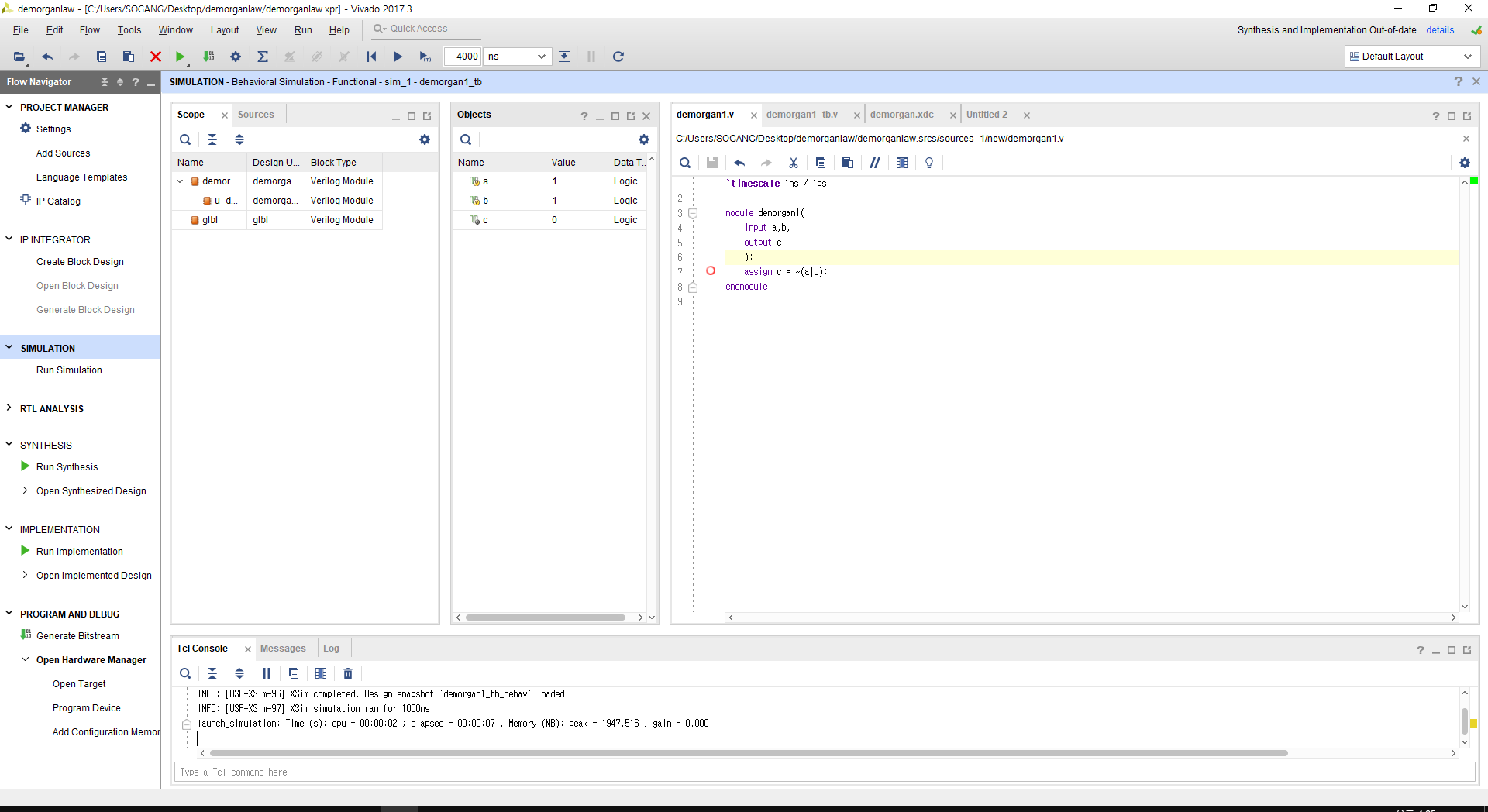
제 1법칙은

(a)는 a+b에 보수를 취한 값이고 (b)는 a의 보수와 b의 보수를 곱한 값이다.

(a)는 ~(a|b)이고 (b)는 ~a & ~b이다.

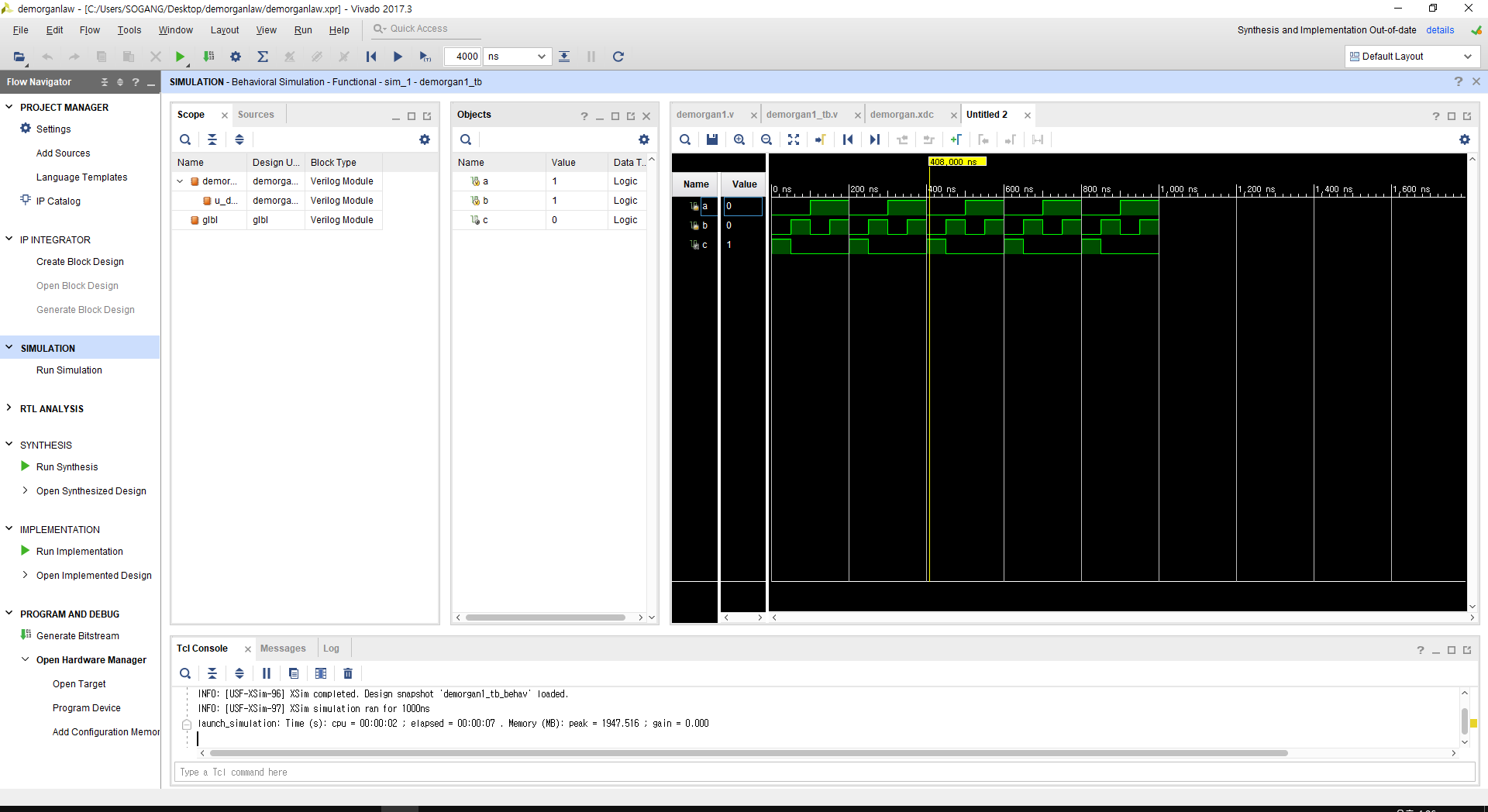
* (a)와 (b)의 Verilog 코딩

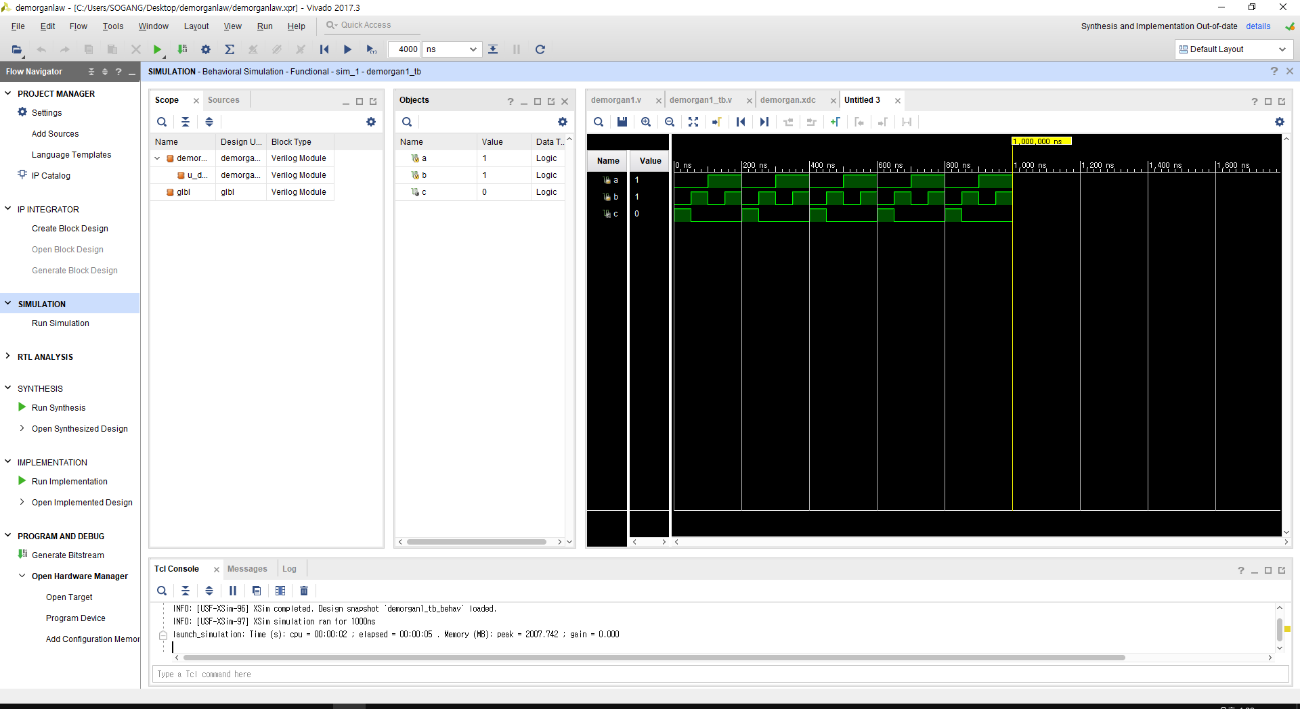
완쪽부터 (a)와 (b)에 해당한다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

* 위쪽부터 (a)와 (b)를 Simulation을 한 결과





위 simulation에서 알 수 있듯이 (a)와 (b)는 서로 같다.

* (a), (b), NOR 비교

(a)는 (a + b)’ 이고, (b)는 a’b’이다. NOR은 a와 b의 or 연산후 인버터를 취해준 값인데, 아 (a), (b), Nor 연산이 서로 같은 것을 진리표를 통해서 살펴보겠다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input a | Input b | NOR(a,b) | (a+b)’ | a’b’ |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |

도표, 라인, 텍스트, 스크린샷이(가) 표시된 사진

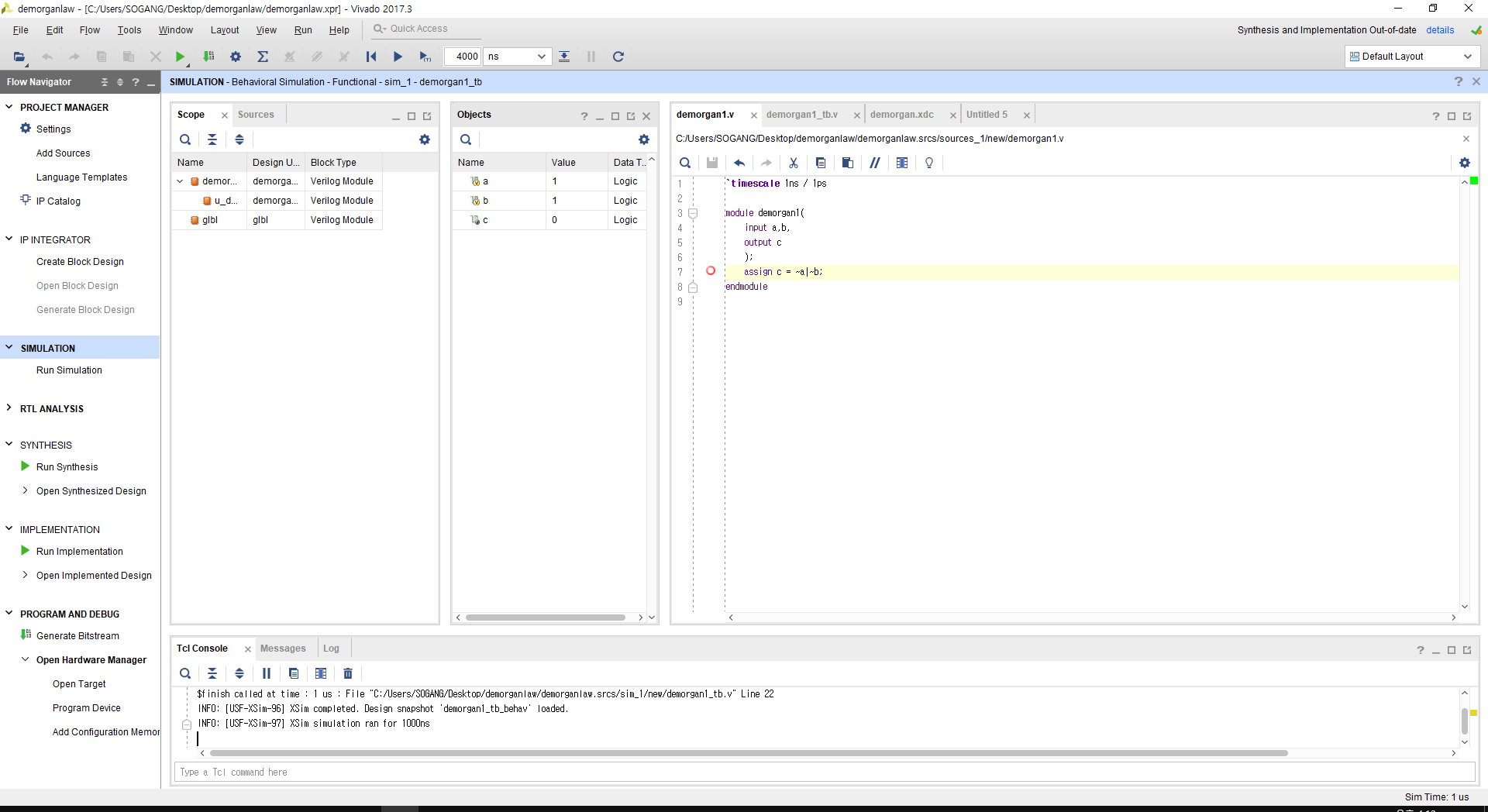
자동 생성된 설명

제2법칙은 ‘A\*B의 보수 취한 것이 A의 보수와 B의 보수와 합한 것과 같다’인데, 이 제2법칙의 (a)는 a와 b의 곱을 보수 취한 것으로 (a)는 ~(a & b)이다. (b)는 a와 b의 보수를 합한 것이니 ~a | ~ b이다.

* Verilog 코딩

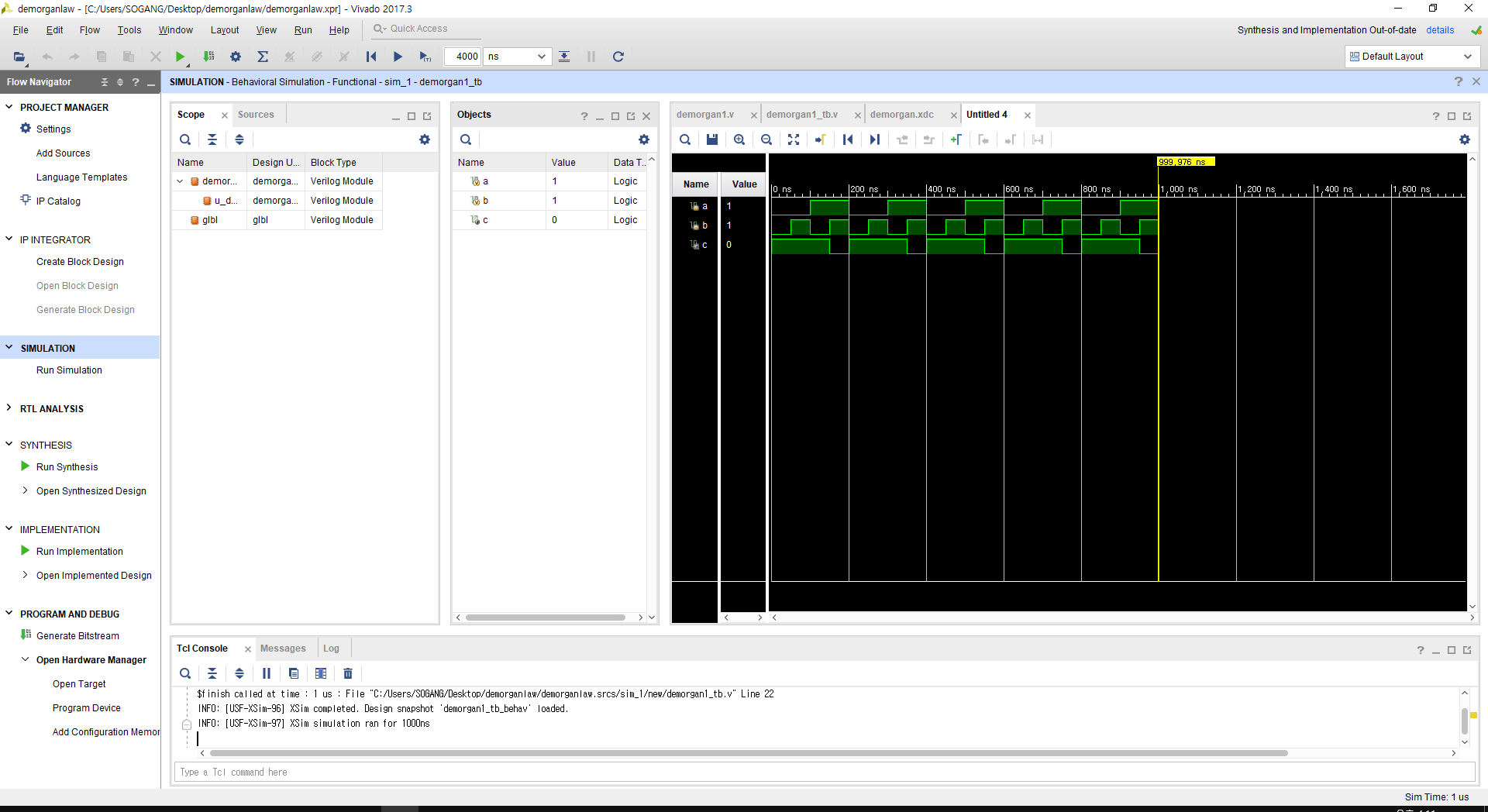
왼쪽은 (a) a와 b의 곱에 보수를 취한 값이고, (b)는 각각 a의 보수 b의 보수를 더한 값이다.

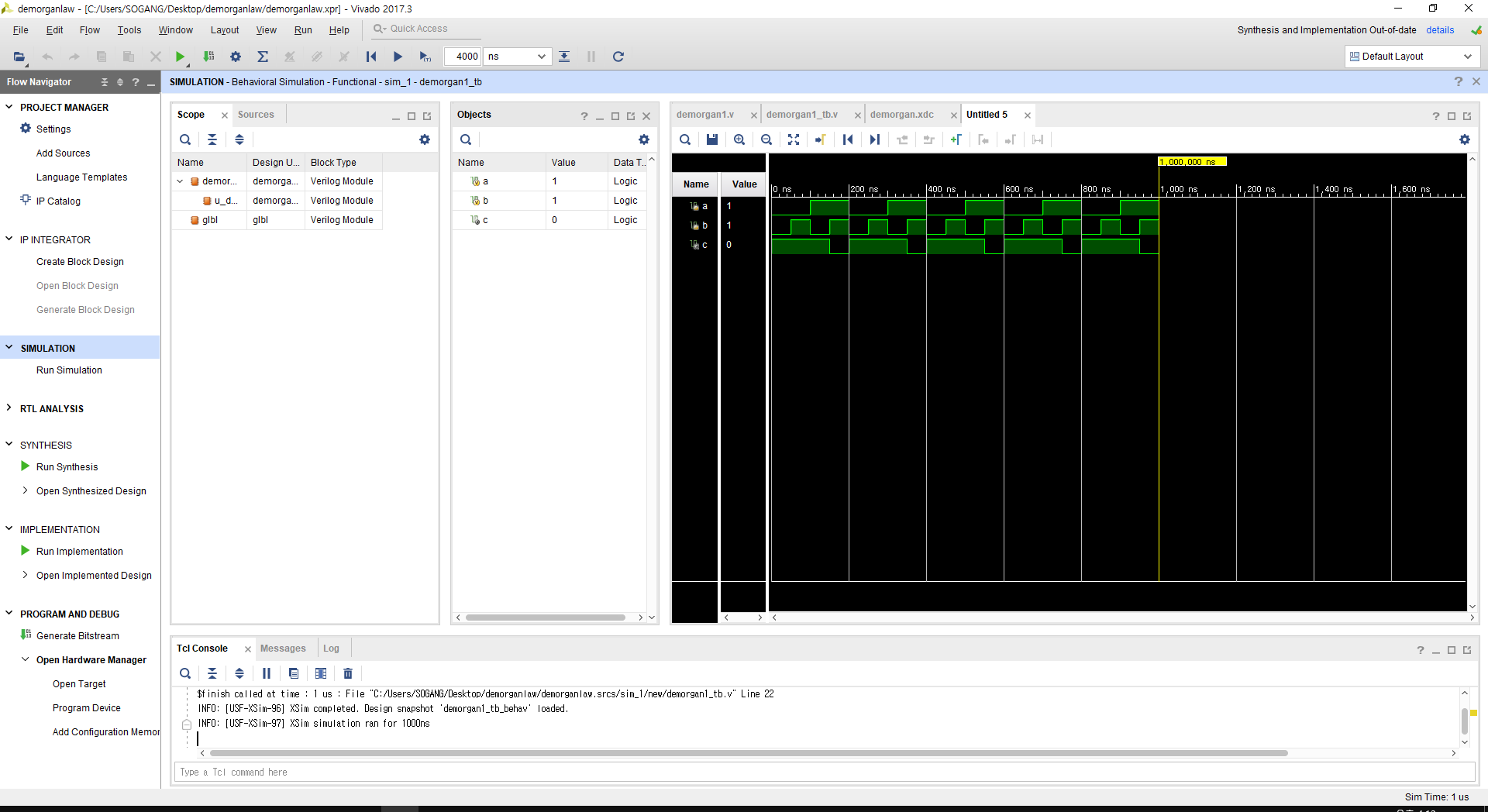
텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

- Simulation 결과

위쪽부터 (a)와 (b)를 시뮬레이션 한 결과이다.

(a)

(b)

제2법칙도 성립함을 알 수 있다.

* (a)와 (b)와 NAND 값 비교

(a)는 (ab)’ 이고, (b)는 a’ + b’이고, NAND는 AND 연산 후 인버터를 씌운 값인데 이를 진리표로 살펴보면 (a)와 (b), NAND는 서로 같음을 알 수 있다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input a | Input b | NAND(a,b) | (ab)’ | a’ + b’ |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |

위 제1법칙과 제2법칙을 simulation 돌릴 때 같은 tb파일을 사용하였는데, tb파일에 대해서 설명하자면

|  |
| --- |
| ‘timescale 1ns /1ps  module demorgan\_tb;  reg a,b;  wire c;  demorgan u\_demorgan(  .a(a), .b(b), .c(c)  );  initial a = 1’b0;  initial b = 1’b0;  always a = #100 ~a;  always b = #50 ~b;  initial begin  #1000  $finish  end  endmodule |

이 tb 파일을 통해 input a, b가 0으로 초기화 되었다가 각각 100, 50 후에 반전된다는 것으로 해석할 수 있다.

드 모르간의 정리 진리표를 작성한다면,

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | A+B | A\*B | A’ | B’ | (A+B)’ | A’B’ | (AB)’ | A’+B’ |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

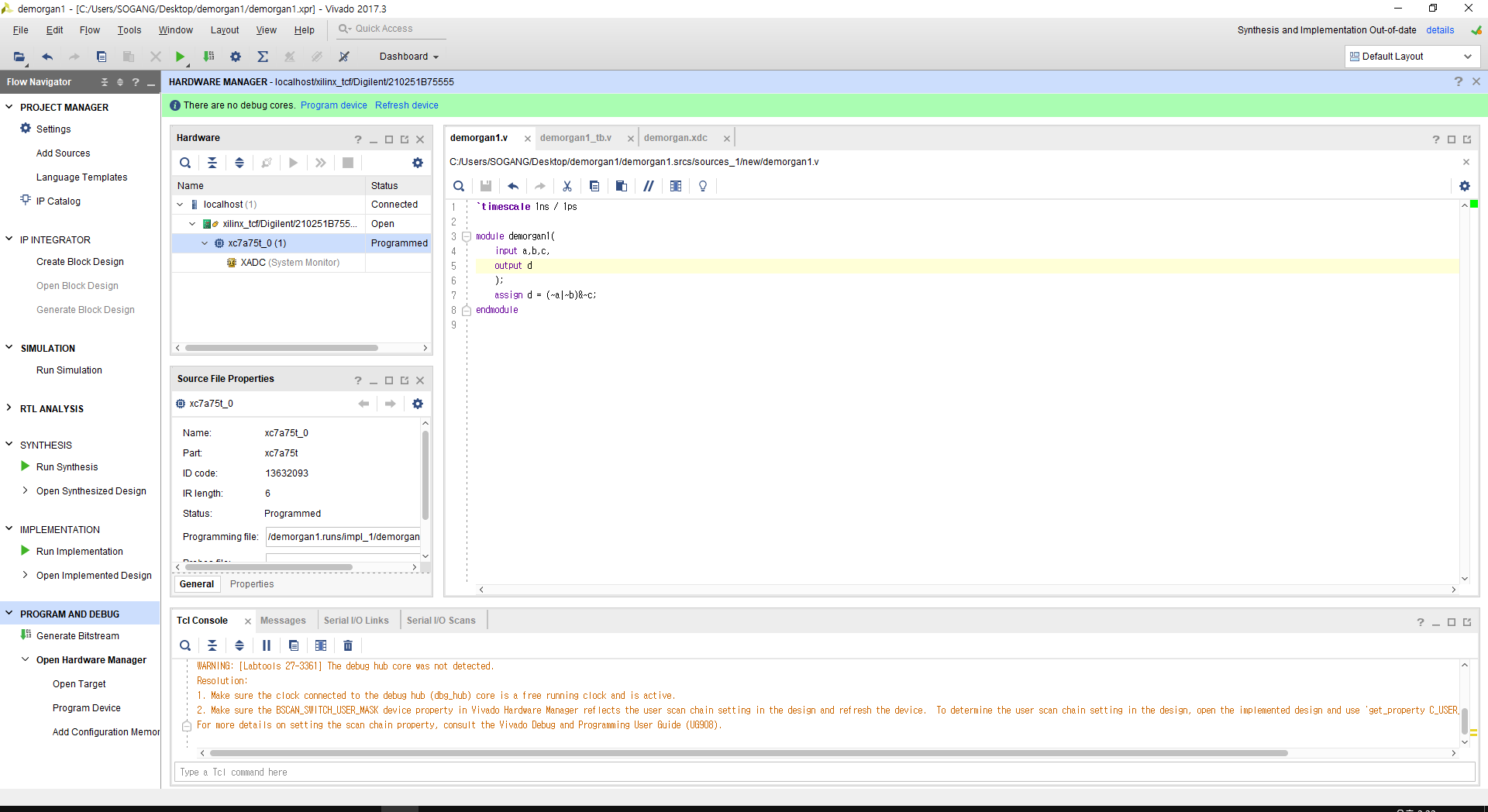
3.

(a)는 (A’ + B’)\*C’이고, (b)는 ((A\*B)+C)’이라고 해보자.

(a)는 논리식으로 (~a | ~b) & ~c이고, (b)는 ~((a & b) | c)이다.

- Verilog 코딩

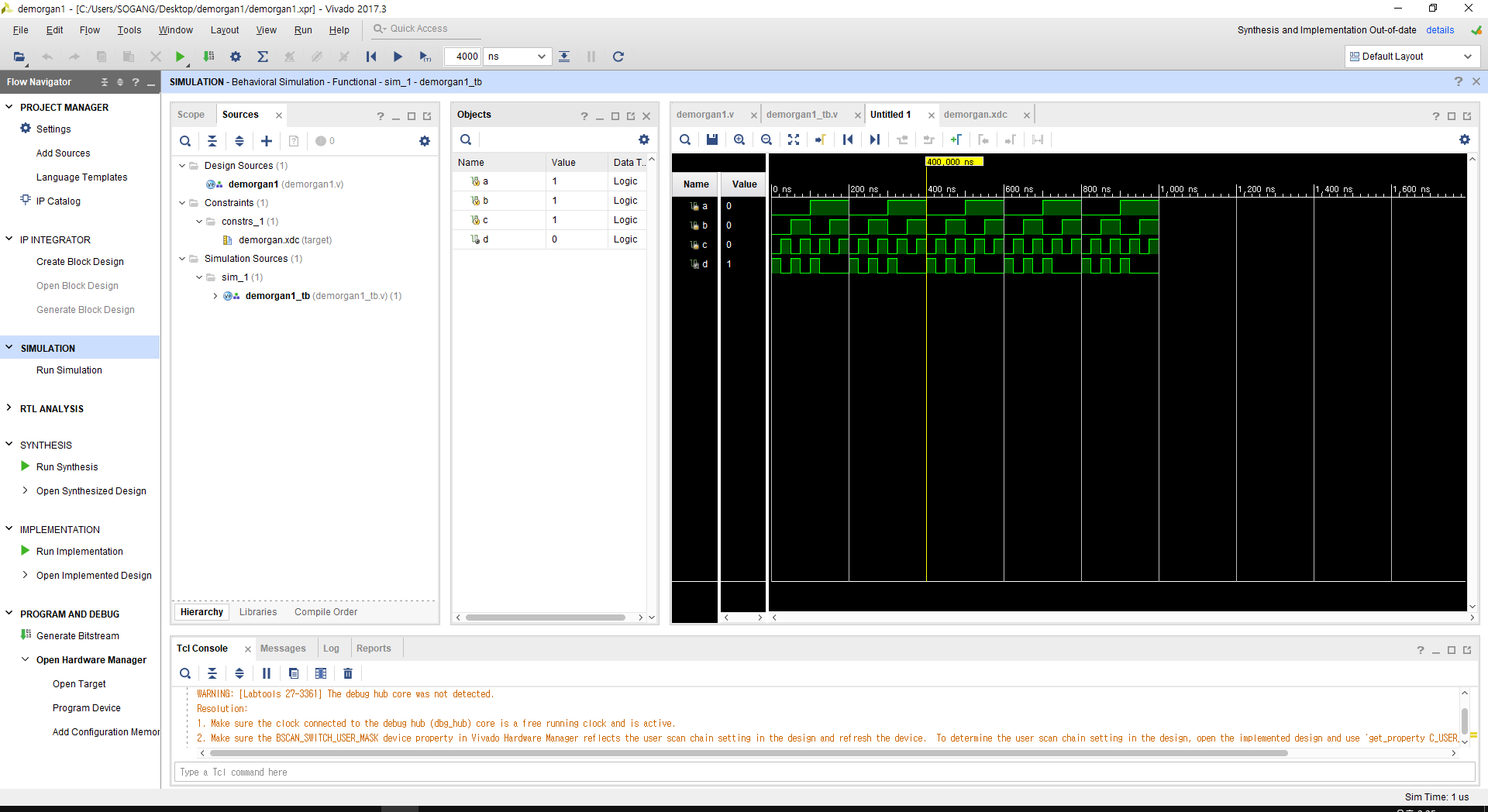
왼쪽에서부터 (a)와 (b)이다.

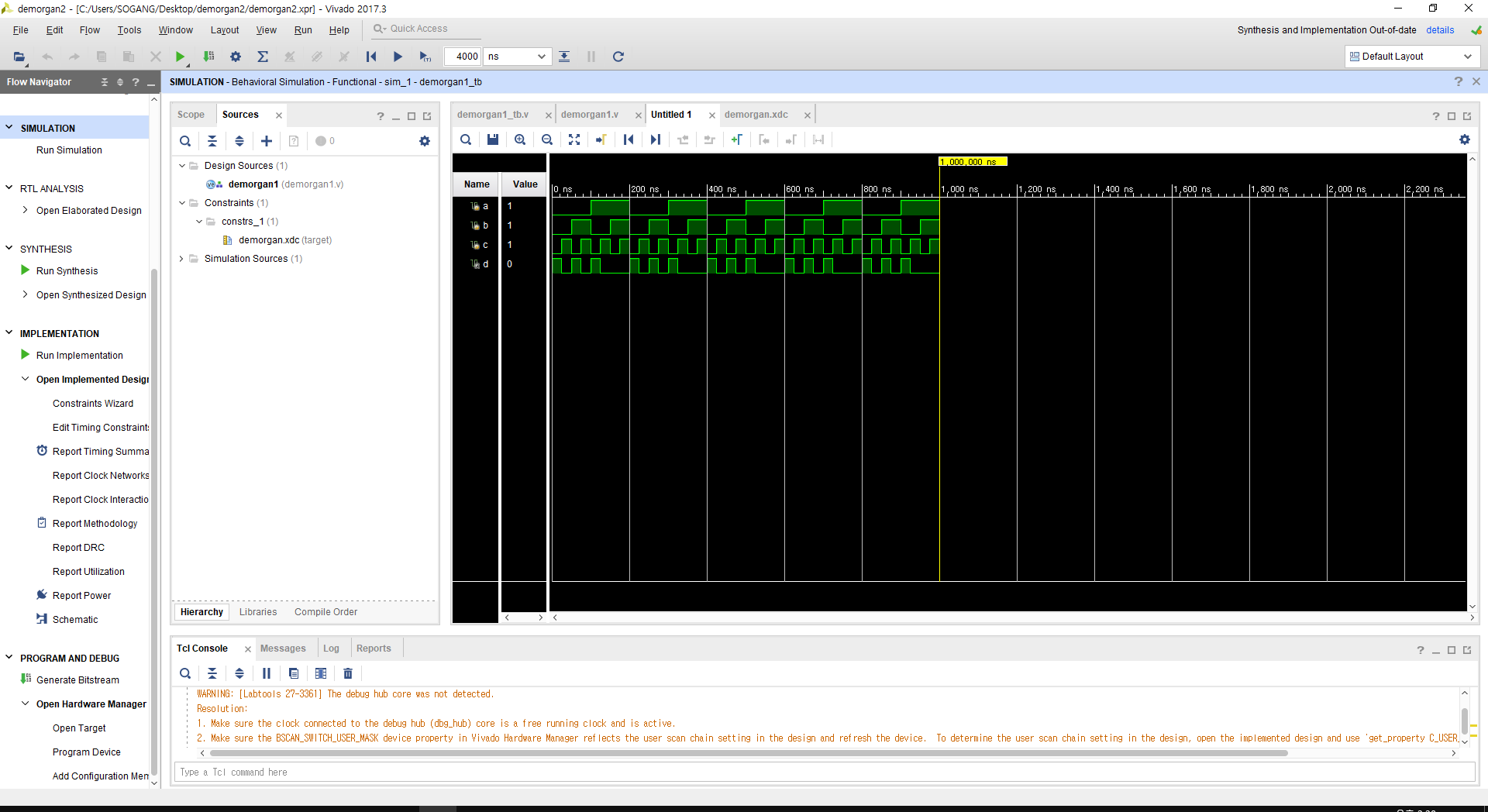
텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* Simulation 결과

위에서부터 (a)와 (b)의 Simulation 결과이다. tb파일을 통해서 해석하는 부분은 3번 항목 마지막에 설명하겠다.





- 진리표 작성

마지막으로 진리표 작성을 해보자면, 위에서부터 (a), (b)의 진리표이다.

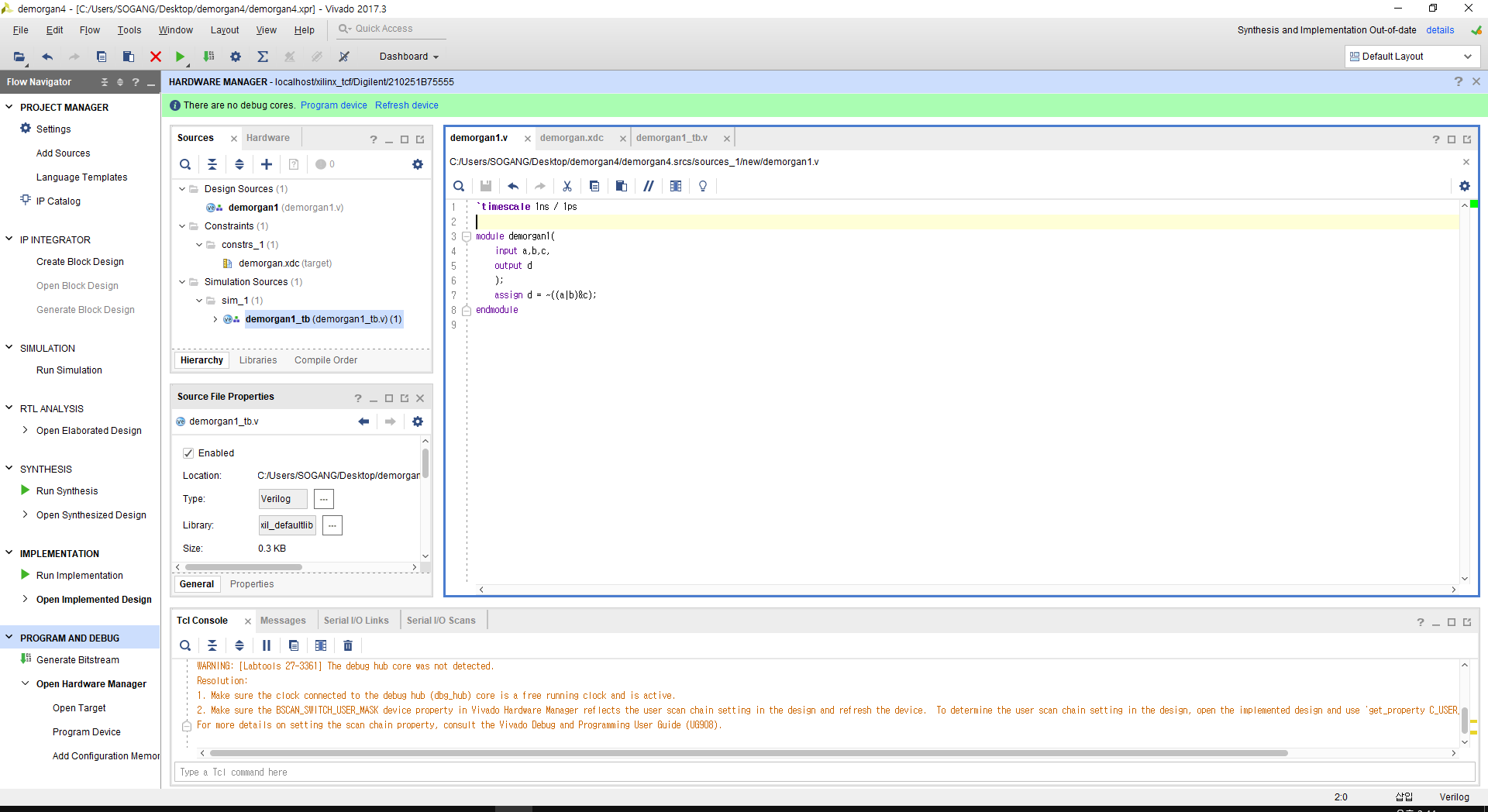
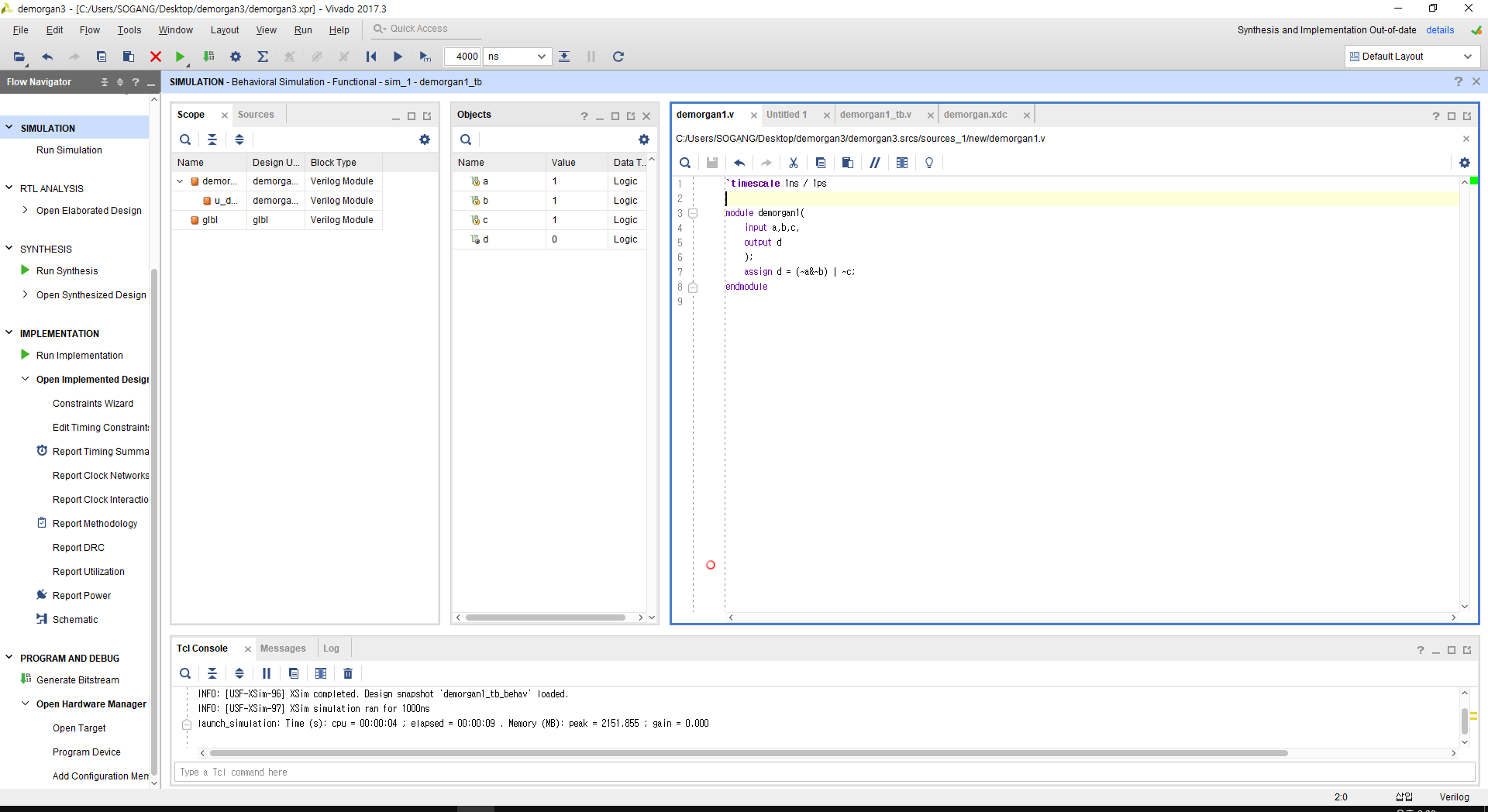
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input A | Input B | Input C | (A’+B’)\*C’ | ((A\*B)+C)’ |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 |

(a)를 (A'\*B')+C' , (b)를 ((A+B)\*C)'라고 하자.

(a)는 논리식으로 (~a & ~b) | ~ c라고 할 수 있고, (b)는 ~((a | b) &c)이다.

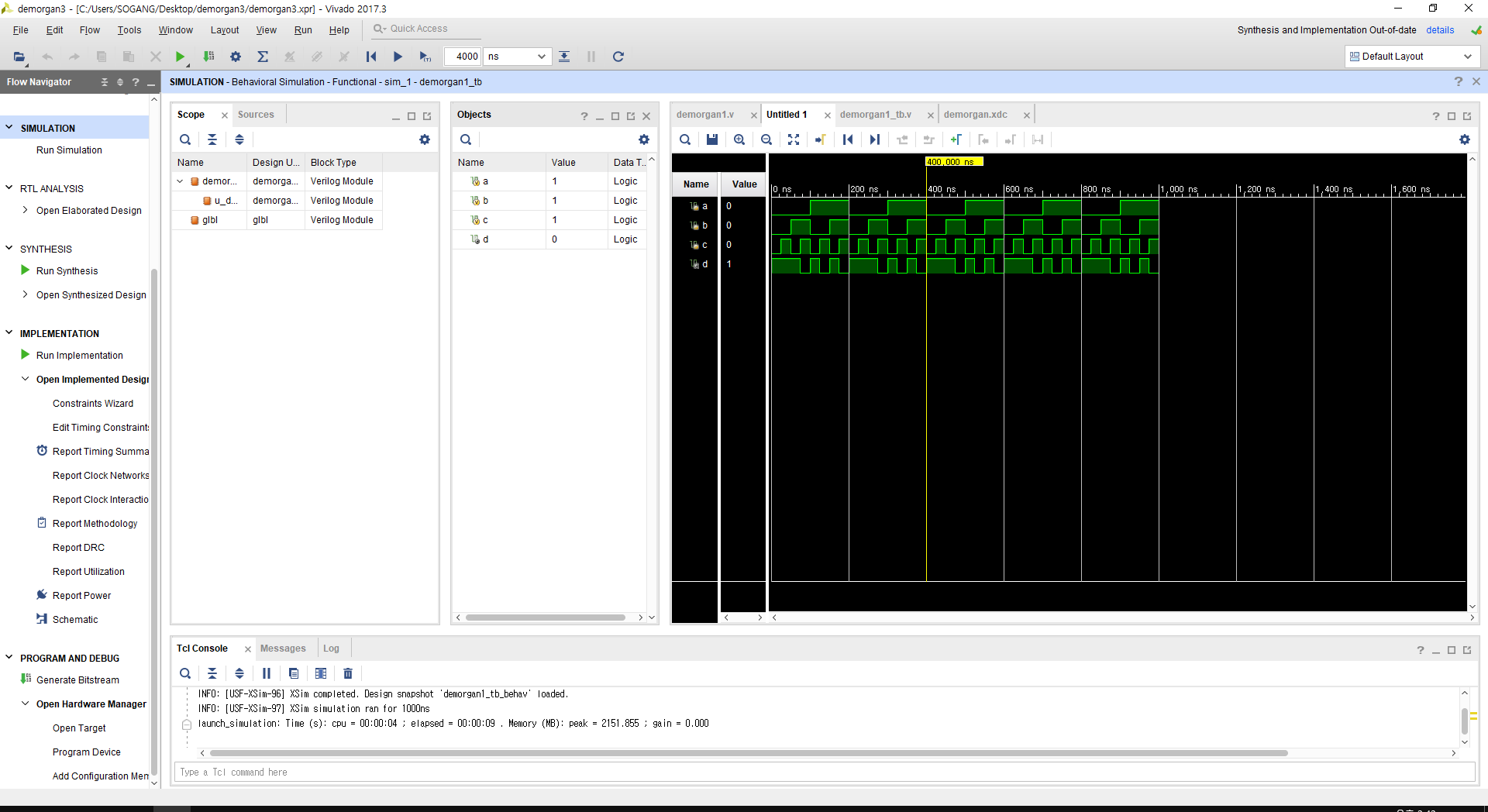
-Verilog 코딩

왼쪽부터 (a)와 (b)에 해당한다.



-Simulation 결과

위에서부터 (a)와 (b)에 해당한다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- 진리표 작성

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input A | Input B | Input C | (A'\*B')+C' | ((A+B)\*C)’ |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |

마지막으로 tb파일을 분석하자면

|  |
| --- |
| ‘timescale 1ns /1ps  module demorgan\_tb;  reg a,b,c;  wire d;  demorgan u\_demorgan(  .a(a), .b(b), .c(c),  .d(d)  );  initial a = 1’b0;  initial b = 1’b0;  initial c = 1’b0;  always a = #100 ~a;  always b = #50 ~b;  always c = #25 ~c;  initial begin  #1000  $finish  end  endmodule |

Input a, b, c가 0으로 처음에 초기화가 되고, 100, 50, 25 이후에 반전되는 것으로 알 수 있다.

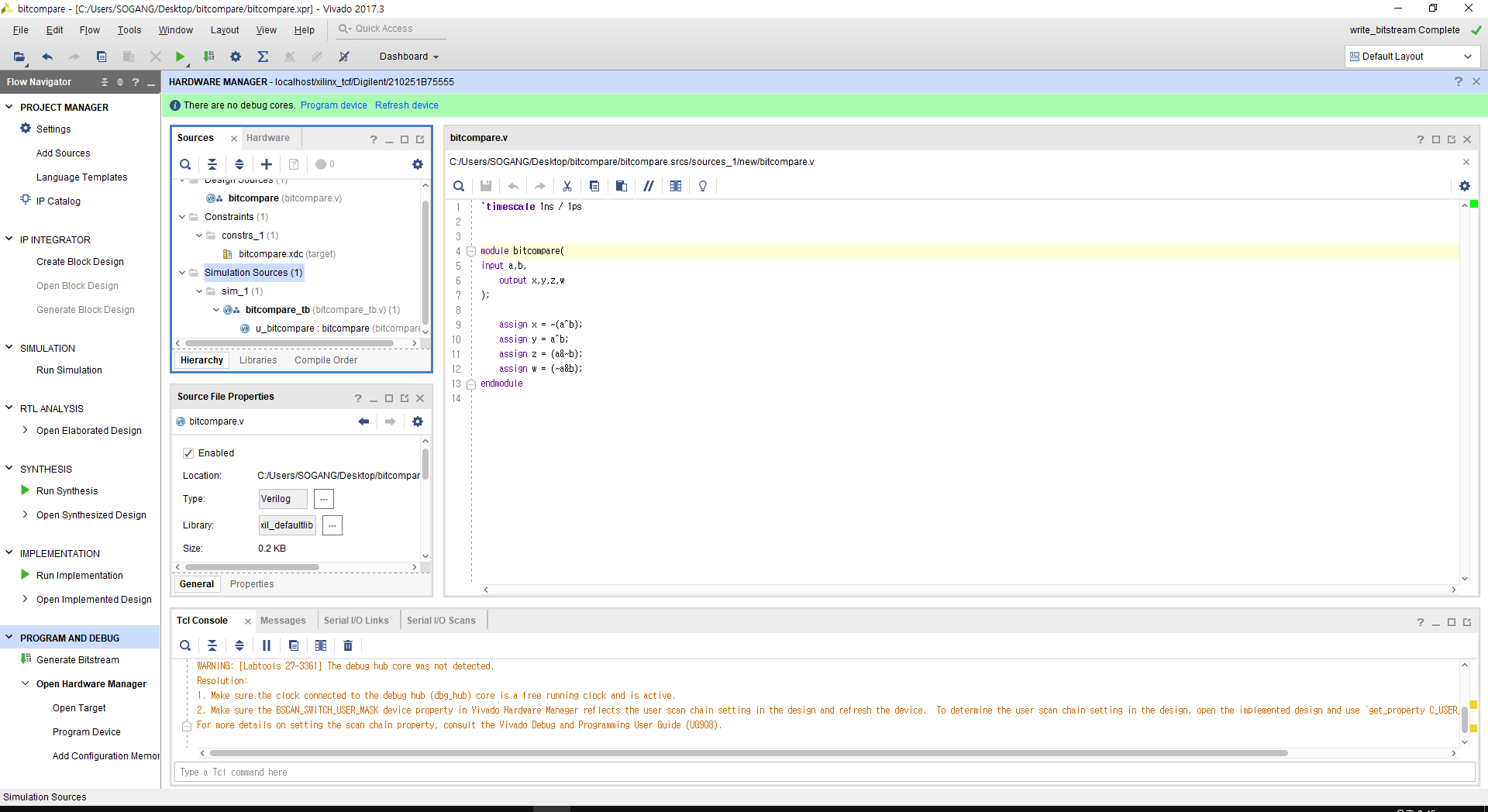
4.

-Verilog 코딩

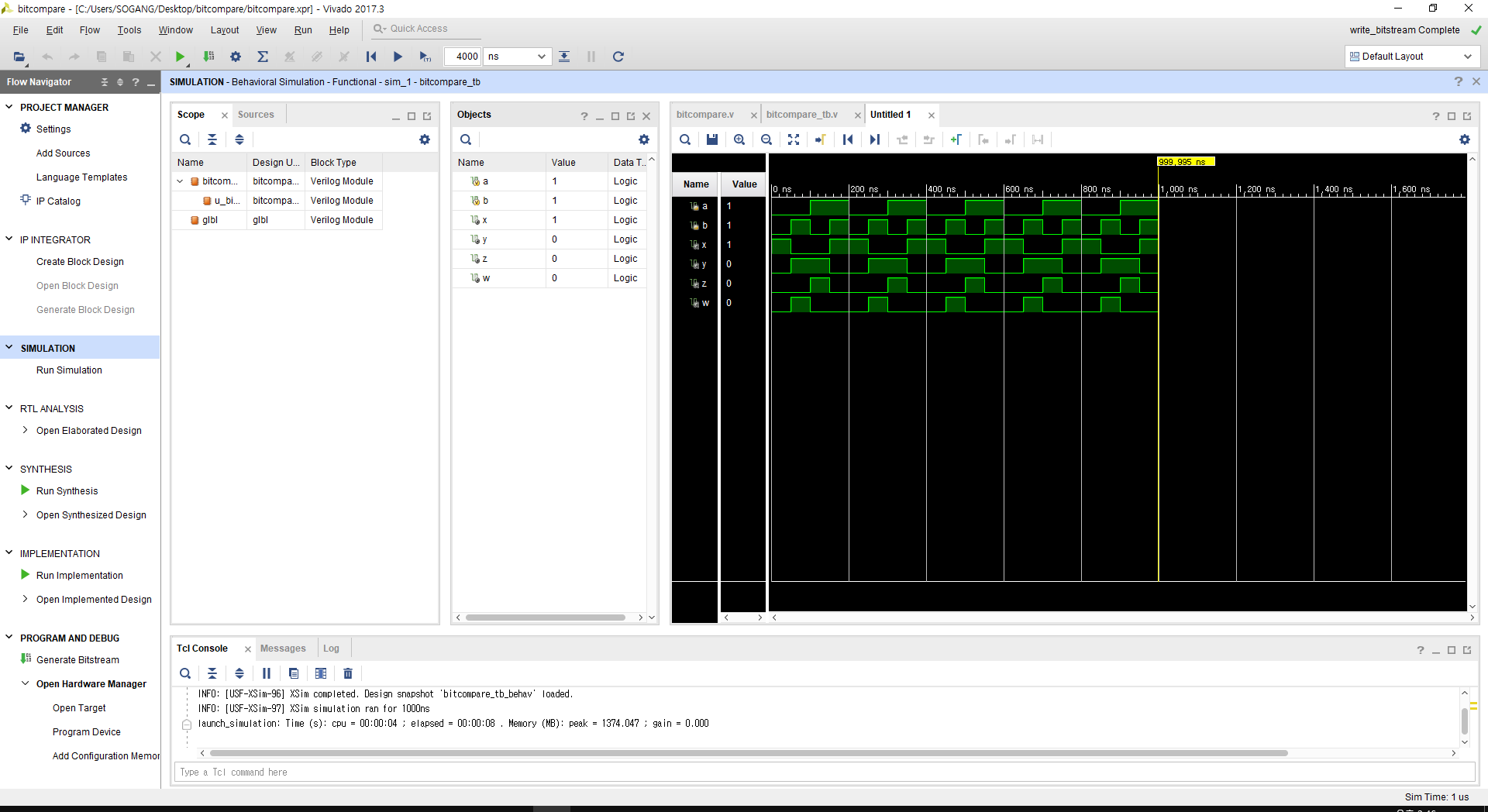
1bit 비교기 진리표를 이용하여, Output x, y, z, w의 논리적 수식을 구현할 수 있는데

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | A=B | A != B | A > B | A < B |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |

A != B의 경우 A와 B의 값이 같지 않아야 1을 출력하고, A= B의 경우에는 값이 같아야 1을 출력하기 때문에 XOR, XNOR으로 이해할 수 있다. 그리고 A > B, A < B는 각각 B의 보수와 A의 곱, A의보수와 B의 곱으로 이해할 수 있다.



* Simulation 결과



5.

NOR 게이트를 논리 수식으로 표현하자면 (A + B)’이고, NAND 게이트를 논리 수식으로 표현하자면 (A \* B)’인데, 이 두가지 모두 각각 드 모르간 정리에 의해서 A’B”, A’ + B’와 동일하다는 것을 증명했다. 또한, 실습에서 진행한 (A’ + B’) \* C’와 (A’ \* B’) + C’를 각각 드 모르간 법칙에 의해서 ((A\*B) +C)’, ((A + B) \* C)’로 정리할 수 있고, 논리적 간소화를 이룰 수 있다. 또한 =. !=, >, <을 진리표를 통하여 논리적 수식으로 재구성할 수 있다.

6.

1 bit 비교기 뿐만 아니라 2 bit 비교기도 존재한다. 1 bit 비교기는 X에 대하여, Y에 대하여만 생각하면 되지만, 2 -bit 비교기는 X1X2, Y1Y2를 생각해야 한다. 다음은 2 bit 비교기에 따른 진리표이다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | Y | X=Y | X!=Y | X>Y | X<Y |
| 00 | 00 | 1 | 0 | 0 | 0 |
| 01 | 0 | 1 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 |
| 11 | 0 | 1 | 0 | 1 |
| 01 | 00 | 0 | 1 | 1 | 0 |
| 01 | 1 | 0 | 0 | 0 |
| 10 | 0 | 1 | 0 | 1 |
| 11 | 0 | 1 | 0 | 1 |
| 10 | 00 | 0 | 1 | 1 | 0 |
| 01 | 0 | 1 | 1 | 0 |
| 10 | 1 | 0 | 0 | 0 |
| 11 | 0 | 1 | 0 | 1 |
| 11 | 00 | 0 | 1 | 1 | 0 |
| 01 | 0 | 1 | 1 | 0 |
| 10 | 0 | 1 | 1 | 0 |
| 11 | 1 | 0 | 0 | 0 |