6주차 결과보고서

전공: 영미어문/컴퓨터공학과 학년: 3학년 학번: 20200185 이름: 박정주

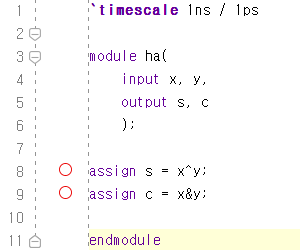
**1.**

6주차 실험의 목적은 다음과 같다. 가산기와 감산기의 개념을 이해하고, Verilog를 통해 반가산기, 전가산기, 반감산기, 전감산기를 구현하고 FPGA에서 감산기와 가산기의 동작을 확인한다. code와 code converter의 개념을 이해하고, Verilog를 통해 8421 to 2421 code converter를 구현한 뒤 FPGA를 통해 converter의 동작을 확인한다.

**2.**

**1) Half Adder(반가산기)**

반가산기의 design source를 다음과 같이 코딩하였다. 두 1bit 수 x, y를 입력으로 받는다. 두 수의 합을 의미하는 s와 Carry를 의미하는 c를 출력한다. s는 x와 y를 xor연산한 값이다. c는 x와 y를 and연산한 값이다.

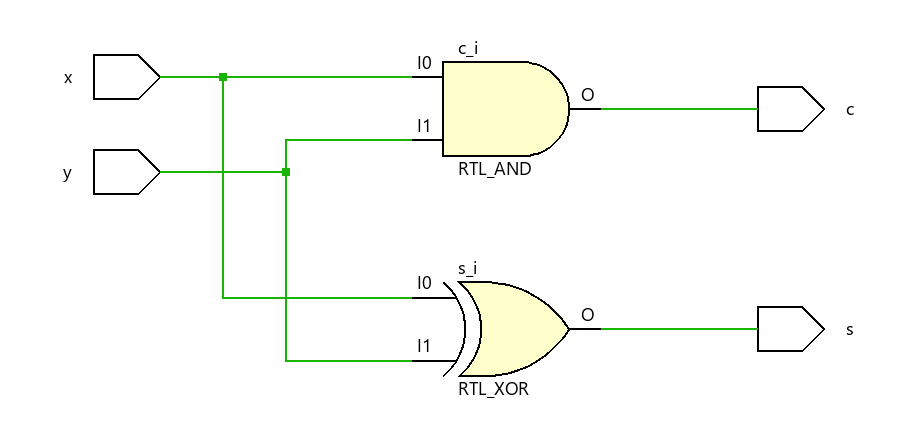


반가산기의 simulation source를 다음과 같이 코딩하였다. initial 구문을 통해 x, y의 값이 0으로 시작하도록 하였다. always 구문을 통해 x는 20 시간단위, y는 30 시간단위마다 값을 부정하도록 하였다.

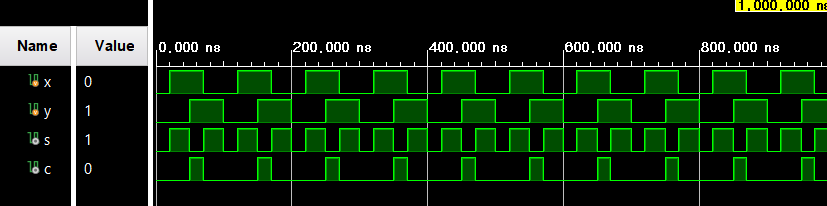
텍스트이(가) 표시된 사진

자동 생성된 설명

반가산기의 schemetic은 다음과 같다.



반가산기의 simulation 결과는 다음과 같다.



실험 결과 반가산기의 진리표는 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| x | y | s | c |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

**2) Full Adder(전가산기)**

전가산기의 design source를 아래와 같이 코딩하였다. 두 1비트 수를 의미하는 x, y와 이전 단계의 Carry를 의미하는 cin을 입력으로 받는다. 두 수의 합을 의미하는 s와 Carry를 의미하는 cout을 출력한다.

assign s = (x^y)^cin;

assign cout = (x&y)|((x^y)&cin);

**텍스트이(가) 표시된 사진

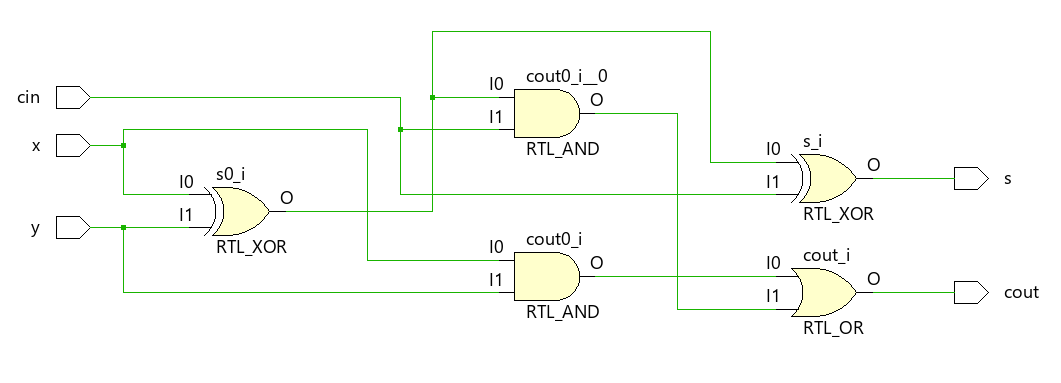
자동 생성된 설명**

전가산기의 simulation source를 다음과 같이 작성하였다. initial 구문을 통해 세 1bit 수 x, y, cin의 값이 모두 0으로 시작하도록 하였다. 3가지 input으로 가능한 모든 경우의 수에 따른 output을 확인할 수 있도록 always문을 작성하였다.

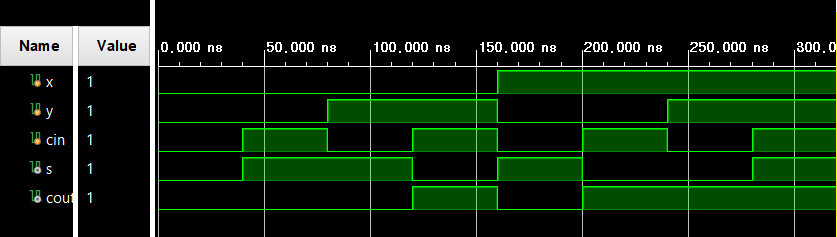
**텍스트이(가) 표시된 사진

자동 생성된 설명**

전가산기의 schemetic은 다음과 같다.

****

전가산기의 simulation 결과는 다음과 같다.



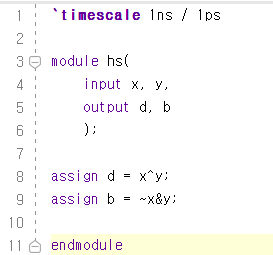
실험 결과 전가산기의 진리표는 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | y | cin | s | cout |
| 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 |

**3.**

**1) Half Subtracter(반감산기)**

반가산기의 design source를 다음과 같이 코딩하였다. 두 1bit 수 x, y를 입력으로 받는다. 두 수의 차를 의미하는 d와 Borrow를 의미하는 b를 출력한다. d는 x와 y를 xor 연산한 값이다. b는 x의 부정과 y를 and 연산한 값이다.

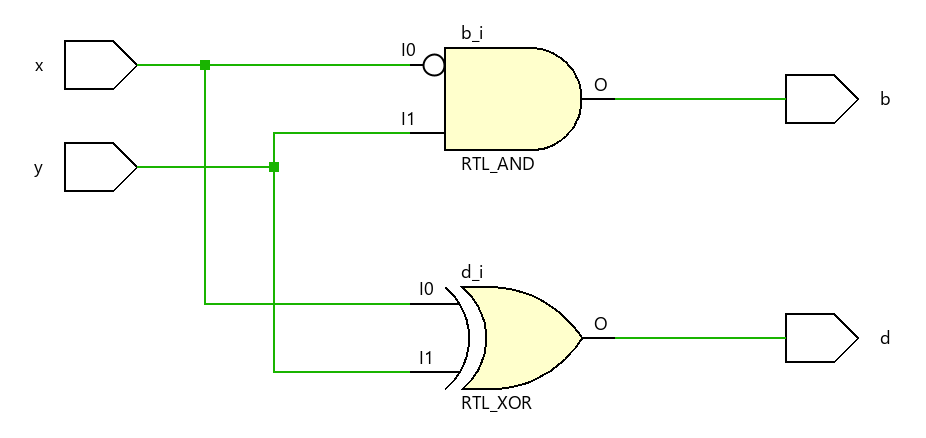


반가산기의 simulation source를 다음과 같이 코딩하였다. initial 구문을 통해 x, y의 값이 0으로 시작하도록 하였다. always 구문을 통해 x는 20 시간단위, y는 30 시간단위마다 값을 부정하도록 하였다.

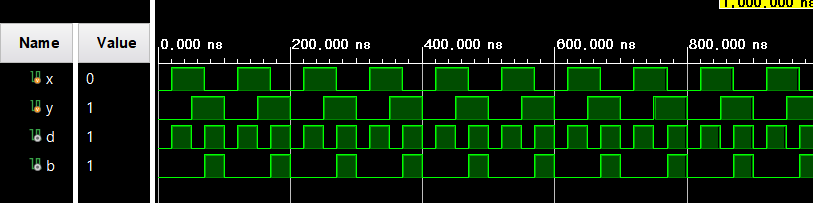
텍스트이(가) 표시된 사진

자동 생성된 설명

반감산기의 schemetic은 다음과 같다.



반감산기의 simulation 결과는 다음과 같다.



실험 결과 반감산기의 진리표는 다음과 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| x | y | b | d |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

**2) Full Subtracter(전감산기)**

전감산기의 design source를 아래와 같이 코딩하였다. 두 1비트 수를 의미하는 x, y와 이전 단계의 Borrow를 의미하는 bin을 입력으로 받는다. 두 수의 차를 의미하는 d와 Borrow를 의미하는 bout을 출력한다.

assign d = (x^y)^bin;

assign bout = (~x&y)|((~x^y)&bin);

텍스트이(가) 표시된 사진

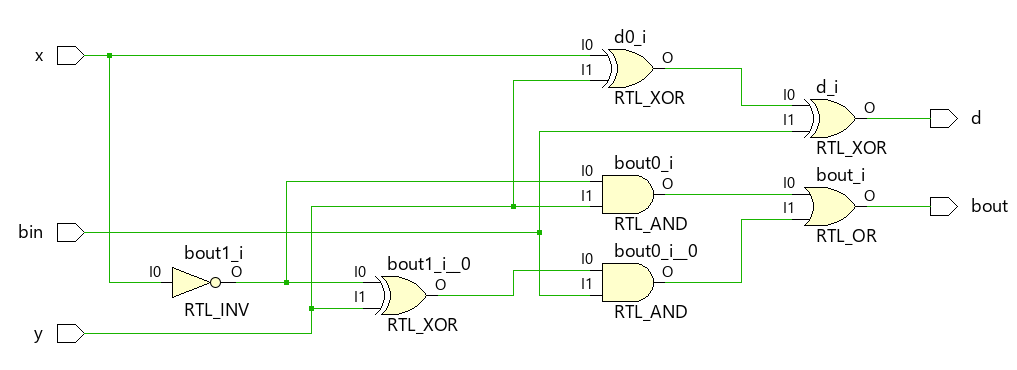
자동 생성된 설명

전감산기의 simulation source를 다음과 같이 작성하였다. initial 구문을 통해 세 1bit 수 x, y, bin의 값이 모두 0으로 시작하도록 하였다. 3가지 input으로 가능한 모든 경우의 수에 따른 output을 확인할 수 있도록 always문을 작성하였다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

전감산기의 schemetic은 다음과 같다.

****

전감산기의 simulation 결과는 다음과 같다.



실험 결과 전감산기의 진리표는 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | y | bin | d | bout |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

**4.**

다음은 순서대로 10진수, 8421 code, 2421 code이다.

0 0000 0000

1 0001 0001

2 0010 0010

3 0011 0011

4 0100 0100

5 0101 1011

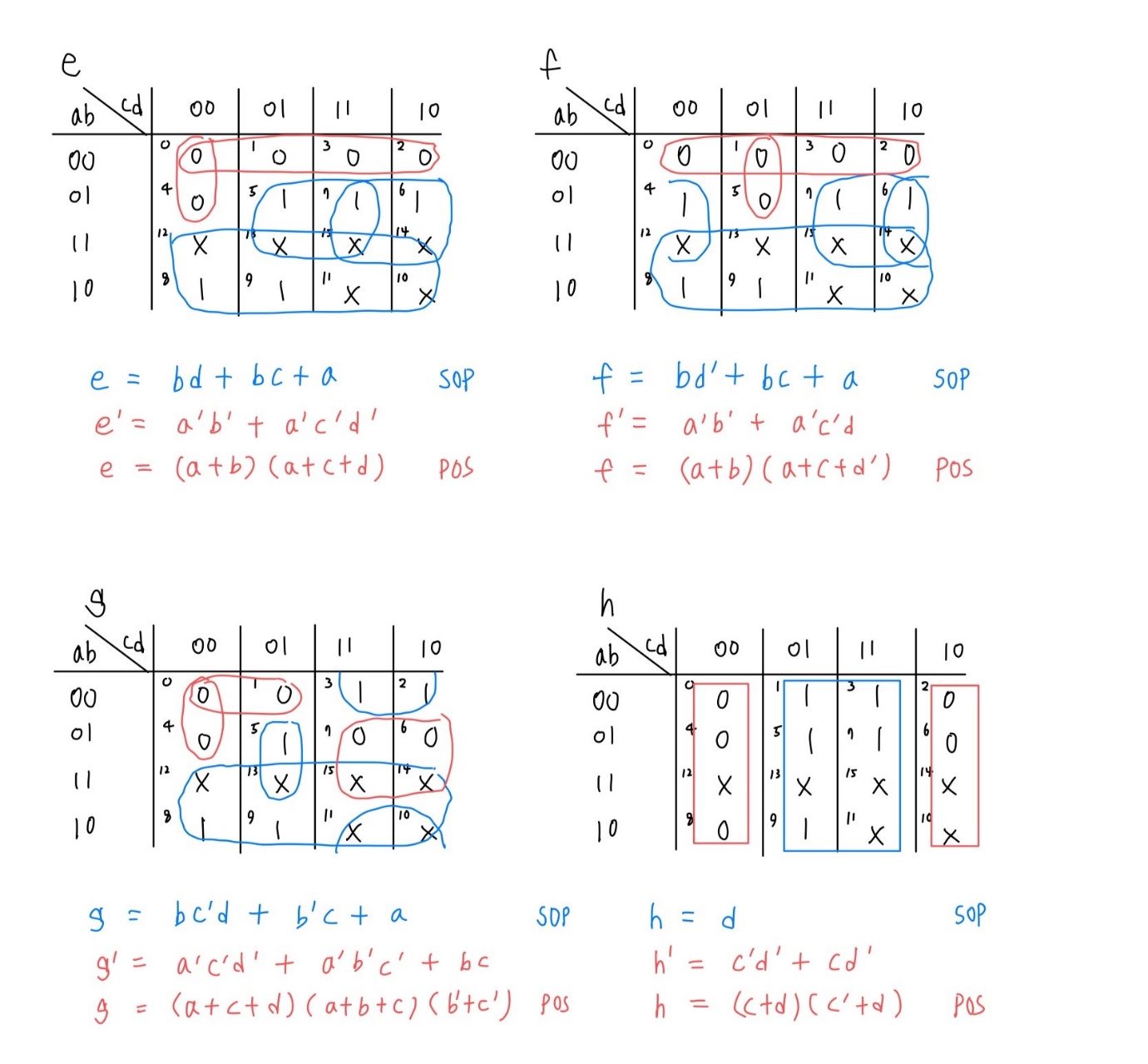
6 0110 1100

7 0111 1101

8 1000 1110

9 1001 1111

이를 참고하여, 8421 코드의 각 자리의 값을 a, b, c, d, 2421 코드의 각 자리의 값을 e, f, g, h로 두었다. 아래와 같이 e, f, g, h의 카르노 맵을 각각 작성하여 SOP form과 POS form을 구하였다.



위에서 구한 e, f, g, h의 SOP form을 최대한 NAND 게이트를 많이 사용하는 형태로 수정하는 과정을 거쳤다.

NOT gate는 다음과 같이 NAND gate 1개로 구현할 수 있다.

~a = ~(a&a)

AND gate는 다음과 같이 NAND gate 3개로 구현할 수 있다.

a&b = ~(~(a&b)&~(a&b))

OR gate는 다음과 같이 NAND gate 3개로 구현할 수 있다.

a|b = ~(~(a&a)&~(b&b))

8421 to 2421 converter의 design source를 다음과 같이 코딩하였다. 8421 code의 각 자리의 값을 의미하는 a, b, c, d를 입력으로 받고, 2421 code의 각 자리의 값을 의미하는 e, f, g, h를 출력한다. NAND gate를 최대한 많이 사용하기 위해 e, f, g를 수정하였다.

**텍스트이(가) 표시된 사진

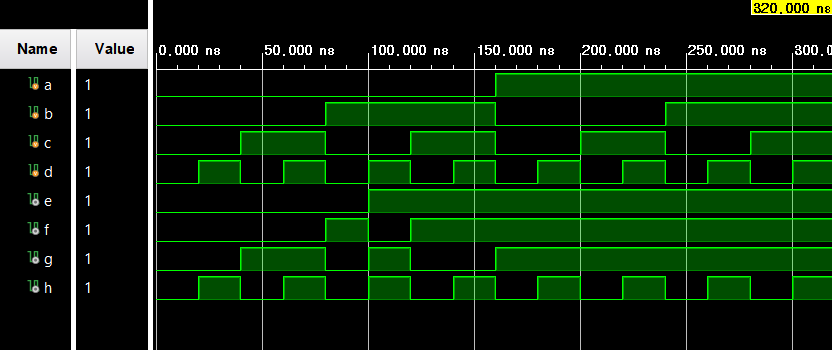
자동 생성된 설명**

0000(0)부터 1001(9)까지의 값이 input으로 입력되었을 때의 결과값을 한눈에 확인하기 위해, simulation source를 다음과 같이 코딩하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

simulation 결과는 다음과 같았다.



8421 code에서 사용하지 않는 6가지의 수를 제외하고 진리표를 작성하면 다음과 같다.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| in a | in b | in c | in d | out e | out f | out g | out h |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

**5.**

결과 검토

반가산기에서는 1bit 수 x, y를 입력 받아 알맞은 s(두 수의 합)와 c(Carry) 값을 출력하였다. 전가산기에서는 1bit 수 x, y와 이전 단계의 Carry인 cin을 입력 받아, 알맞은 s(두 수의 합)와 cout(Carry) 값을 출력하였다. 반감산기에서는 1bit 수 x, y를 입력 받아 알맞은 d(두 수의 차)와 c(Carry) 값을 출력하였다. 전감산기에서는 1bit 수 x, y와 이전 단계의 Borrow인 bin을 입력 받아 알맞은 d(두 수의 차)와 cout(Carry) 값을 출력하였다. 8421 to 2421 converter에서는, 8421 code로 표현된 10진수의 각 자리인 a, b, c, d를 입력 받아, 2421 code의 각 자리인 e, f, g, h를 알맞은 값으로 출력하였다.

논의사항

8421 to 2421 converter의 e, f, g에서는 모두 3 input OR gate가 사용되고 있다.

assign e = ~(~(b&d)&~(b&d))|~(~(b&c)&~(b&c))|a;

assign f = ~(~(b&~(d&d))&~(b&~(d&d)))|~(~(b&c)&~(b&c))|a;

assign g = (b&~c&d)|~(~(~(b&b)&c)&~(~(b&b)&c))|a;

NAND gate만을 사용하여 converter를 구현하기 위해서는 추가적인 수정이 필요할 것으로 보인다.

a|b = ~(~(a&a)&~(b&b))

위와 같이 OR gate를 NAND gate 3개를 사용하여 구현할 수 있으므로,

a|b|c

= (a|b)|c

= ~(~((a|b)&(a|b))&~(c&c))

= ~(~((~(~(a&a)&~(b&b)))&~(~(a&a)&~(b&b)))&~(c&c))

a, b, c를 입력으로 받는 3 input OR gate는 위와 같이 나타낼 수 있을 것이다. 따라서 위의 식을 이용하면 e, f, g를 NAND gate만을 사용하여 구현할 수 있다.

예를 들어, e = ~(~(b&d)&~(b&d))|~(~(b&c)&~(b&c))|a; 이므로,

~(~((~(~(a&a)&~(b&b)))&~(~(a&a)&~(b&b)))&~(c&c)) 이 식의 a에 a를 대입, b에 ~(~(b&d)&~(b&d))를 대입, c에 ~(~(b&c)&~(b&c))를 대입하면 NAND gate만을 사용하여 e를 구현할 수 있다.

**6.**

아래 그림과 같이, NOT gate, AND gate, OR gate, NOR gate, XOR gate, BUFFER gate를 NAND gate만 사용해서 구현할 수 있다.

