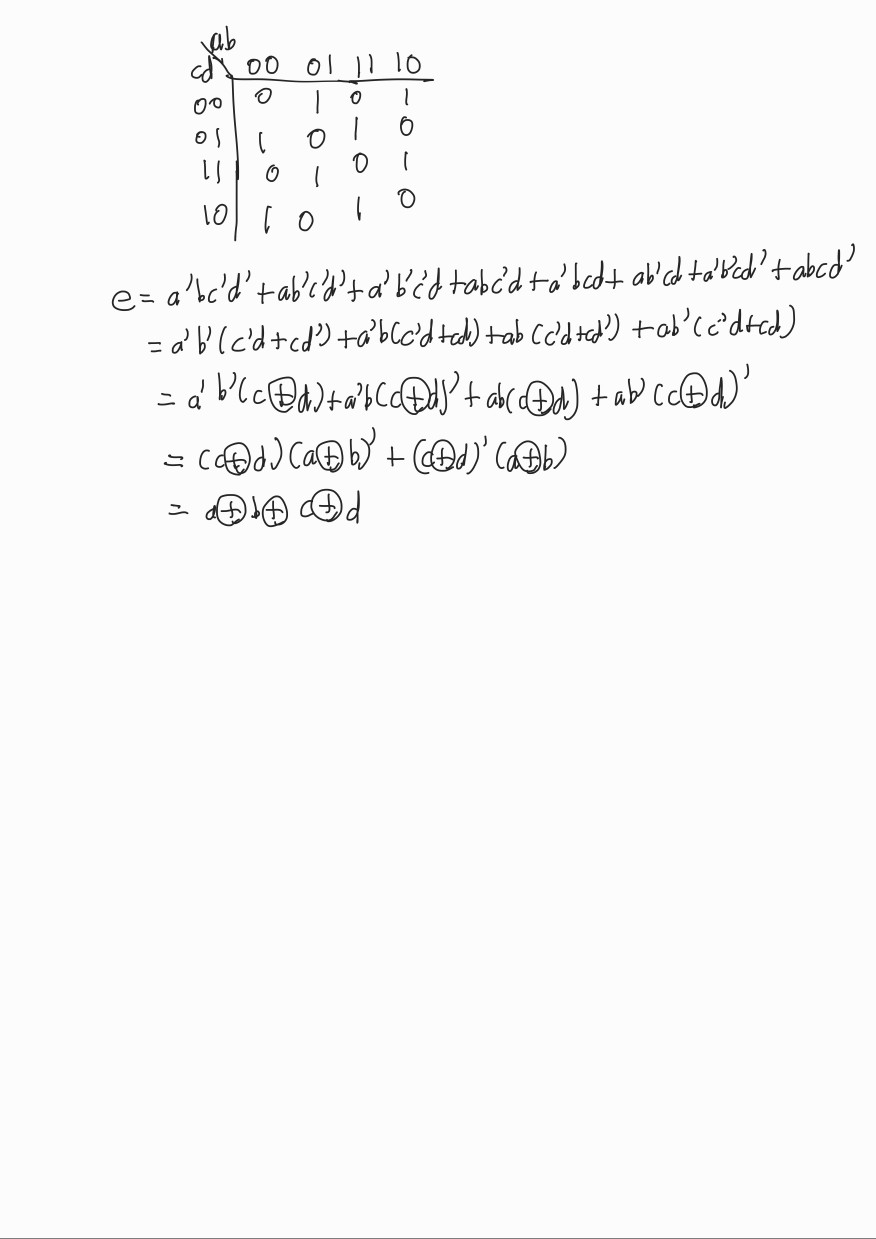
**7주차 결과보고서**

**전공: 아트앤테크놀로지 학년: 3학년 학번: 20191098 이름: 백승주**

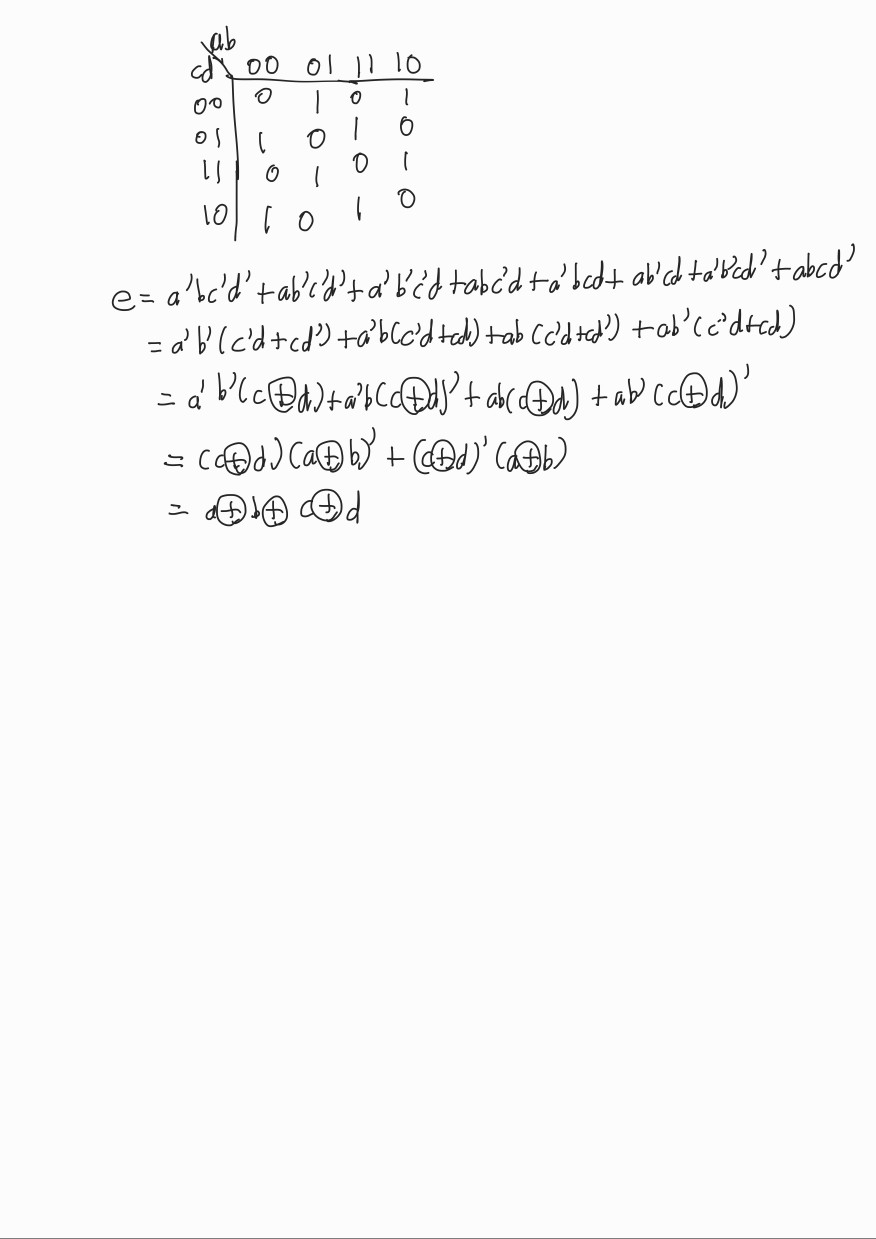
**1-1** even parity bit 생성기는 전송하는 데이터에 포함된 1의 개수를 짝수로 만들어주는 생성기다. 4 개의 input을 가지는 생성기의 진리표를 작성하면 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | b | c | d | e |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

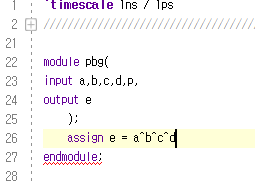
위 진리표를 토대로 카르노 맵을 그리면 다음과 같다.



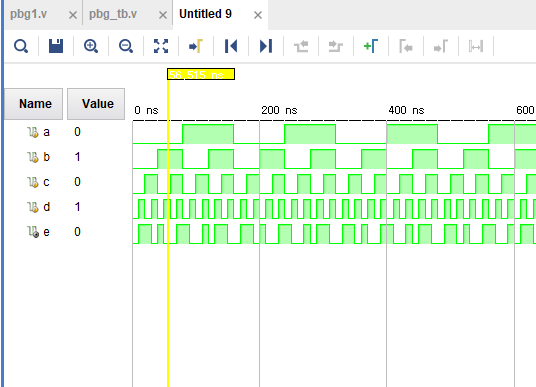
위 카르노 맵의 경우 1이 다 제각각 흩어져 있어 1을 묶어 카르노 맵을 이용해 식을 단순화시키는 것은 더 이상 불가능하다. 카르노 맵을 토대로 다음의 식을 세우고 그 식을 다음과 같이 정리할 수는 있다.



위의 xor식을 Verilog 형태로 나타내면 a^b^c^d 가 된다. 따라서 Verilog 식으로 나타낸 결과는 다음과 같다.



Parity bit 생성기가 생성하는 비트의 이름을 e로 설정했다. 위 코드에 따른 testbench 파일을 작성해 시뮬레이션 결과를 얻으면 다음과 같다.

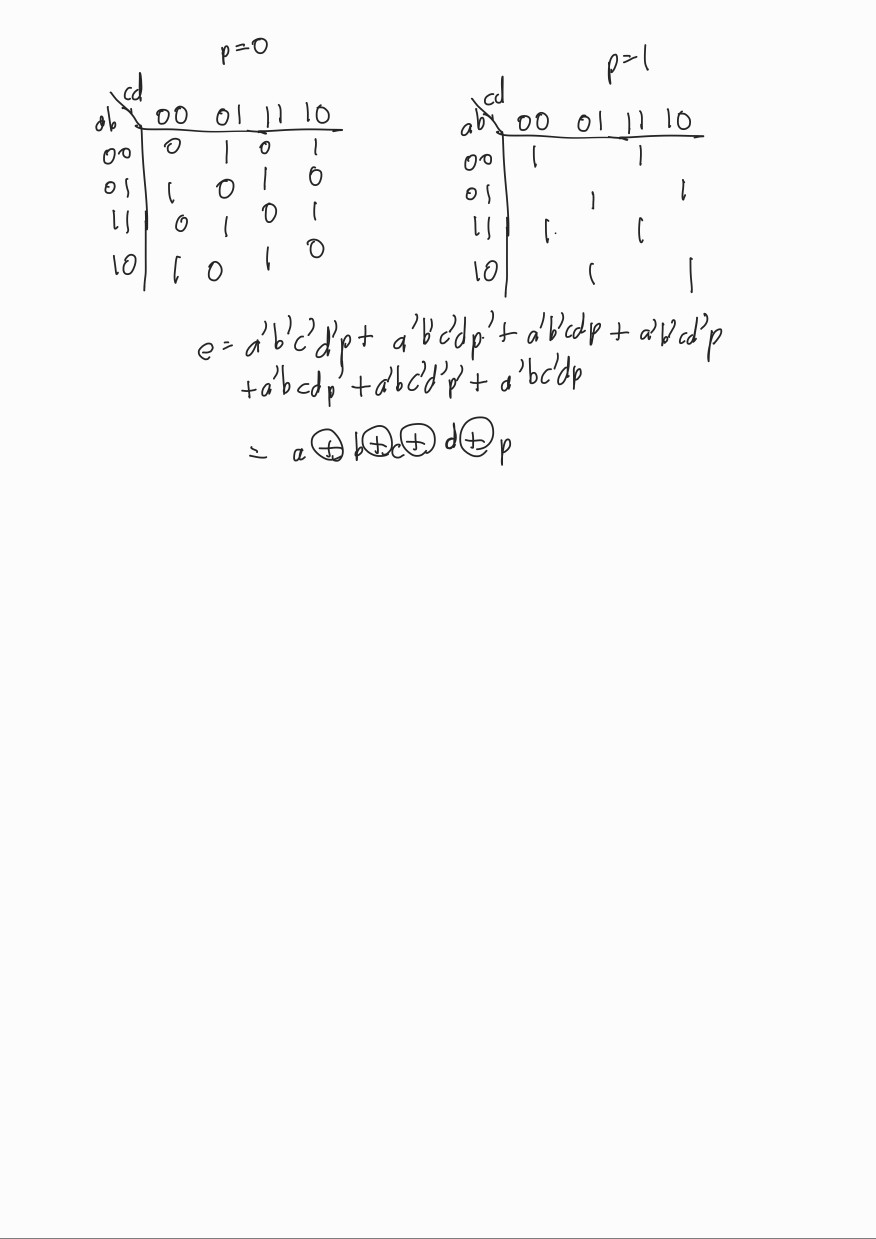


위 시뮬레이션 결과를 보면 의도했던 대로 a=0. B=1, c=0, d=0으로 1이 짝수일 경우 0을 출력해 1을 짝수로 만들어주는 것을 알 수 있다.

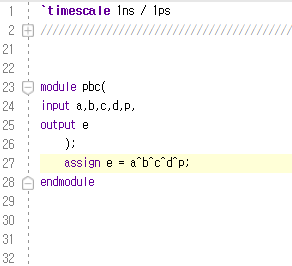
**1-2.** even parity bit 조사기는 받은 데이터의 1의 개수가 짝수인지를 검사해주는 회로다. 만일 1의 개수가 짝수라면 오류가 발생하지 않았으므로 0을 출력하고 1의 개수가 홀수라면 오류가 발생하므로 1을 출력해야 된다. 그에 따른 진리표는 다음과 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a | b | c | d | p | p |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

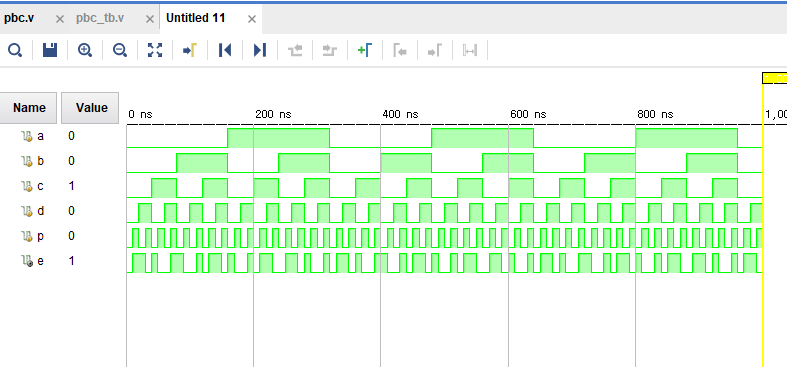
이 진리표에 따른 카르노 맵은 다음과 같다. P=0일 때와 p=1일 때 각각의 경우를 나눠서 카르노 맵을 그리고 식을 만들었다.



이에 따른 Verilog 코드는 다음과 같다.



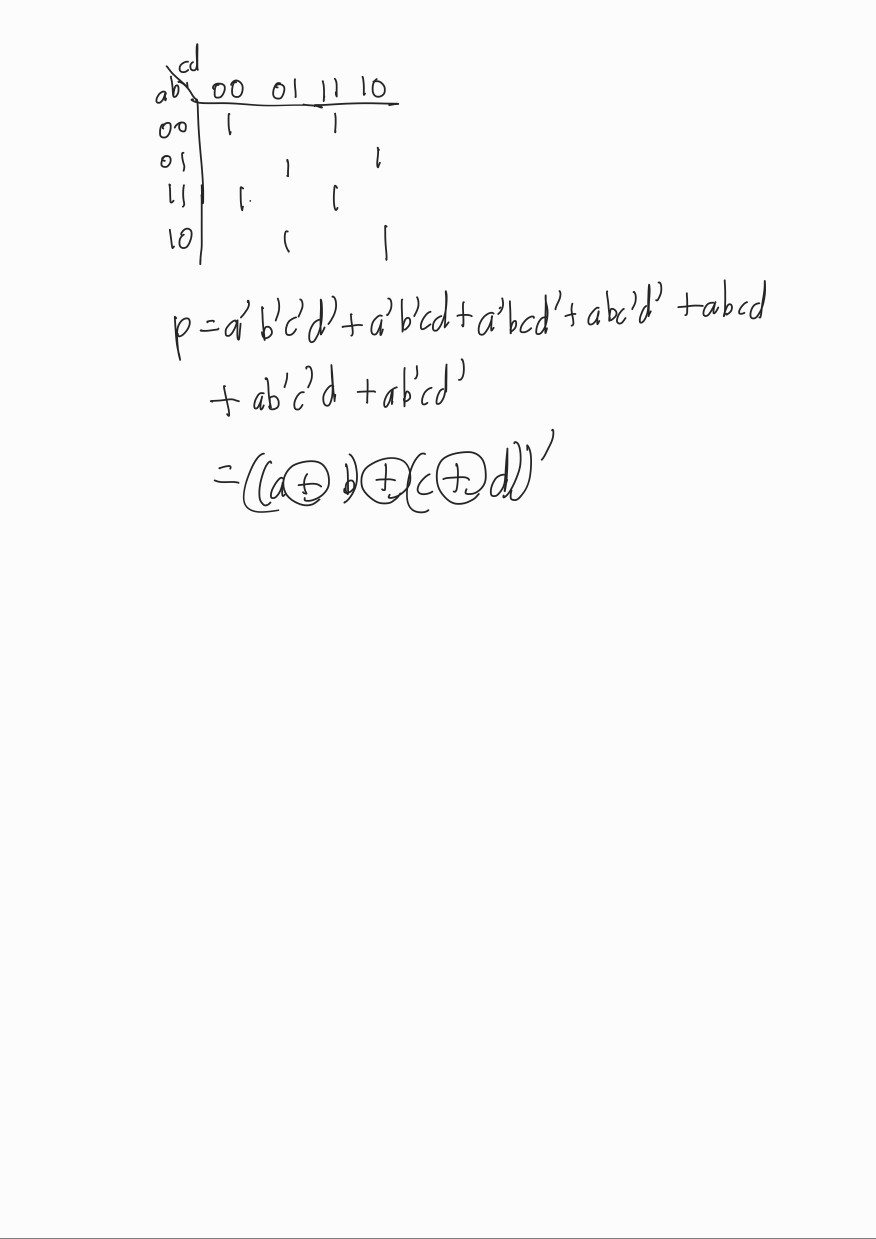
이 Verilog 코드를 토대로 시뮬레이션을 돌리면 다음과 같은 결과가 나온다.



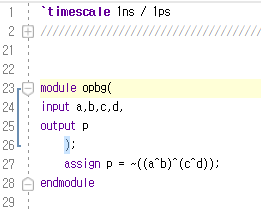
의도한 대로 a=0, b=0, c=1, d=0, p=0 으로 1의 개수가 홀수일 경우 오류가 발생했다는 뜻으로 1을 출력하는 것을 알 수 있다.

**2-1.** odd parity bit 생성기는 전송하는 데이터의 1의 개수를 홀수로 만들어주는 parity bit를 생성하는 회로다. 4개의 input에 따른 생성기의 진리표를 생성하면 다음과 같다.

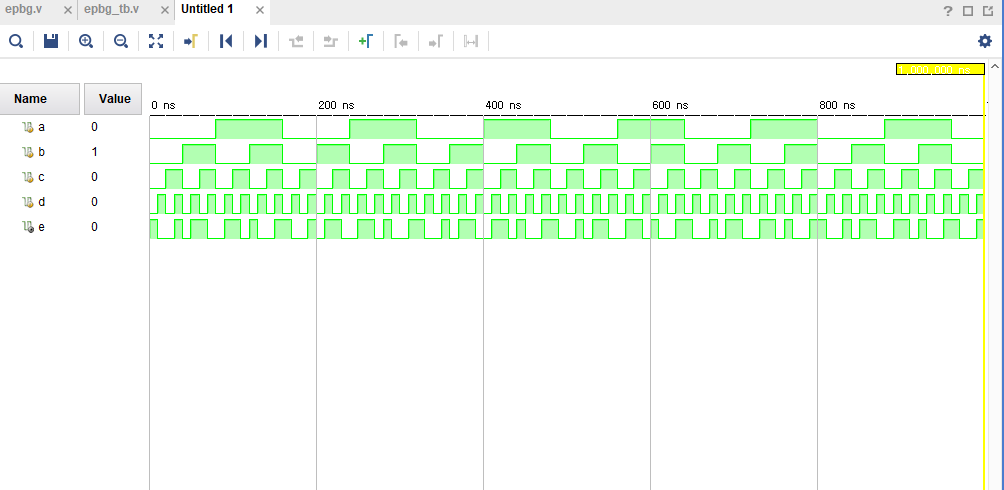
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | b | c | d | p |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |



위 카르노 맵을 통한 식을 토대로 Verilog 코드를 작성하면 다음과 같다.



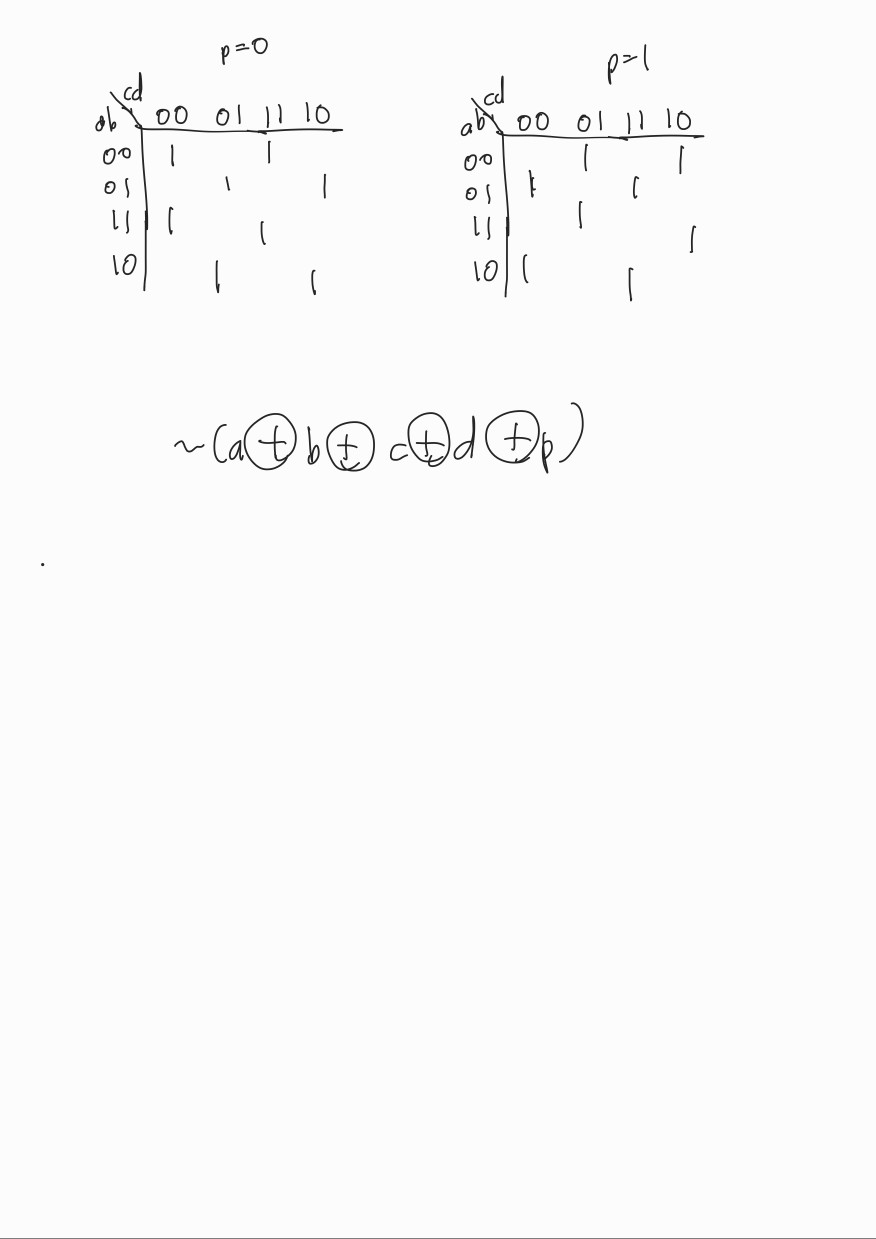
위 코드에 따른 시뮬레이션 결과는 다음과 같다.



**2-2.** odd parity bit 조사기는 받은 데이터의 1의 개수가 홀수 개가 맞는지 검사해주는 회로다. 이 회로는 만일 데이터의 1의 개수가 홀수라면 문제가 없으므로 0을 출력하고 짝수 개라면 오류가 발생했다는 것이므로 1을 출력한다. 이에 따른 진리표는 다음과 같다.

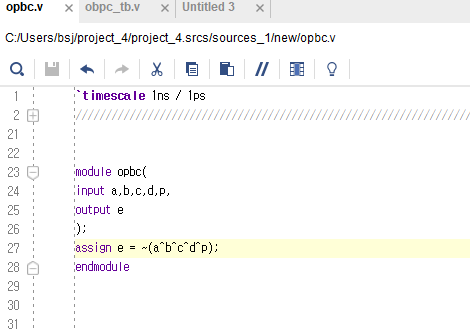
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a | b | c | d | p | e |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

진리표를 토대로 카르노 맵을 그리면 다음과 같다.



이 카르노 맵은 even parity bit 점검기의 반대이므로 ~(a^b^c^d^p)나 다름없다.

위 식을 토대로 Verilog 소스 코드를 작성하면 다음과 같다.



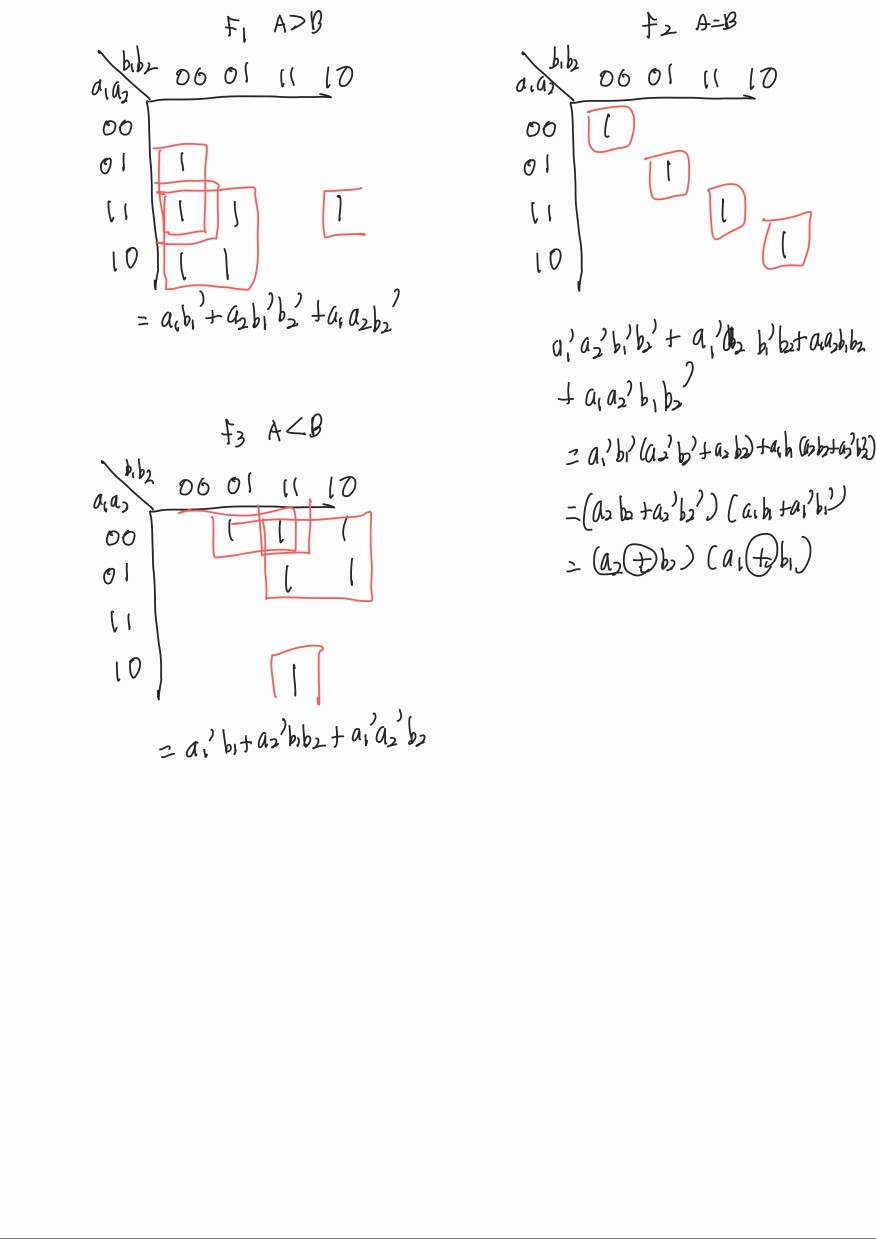
위 식을 토대로 시뮬레이션 결과를 도출하면 다음과 같다.



**3.** 2 bit 이진 비교기는 2 bit 크기의 이진수 a와 b를 입력 받고 두 이진수를 비교하며 만일 a>b 비교했을 때 성립하면 1을 출력한다. 2 bit 크기의 두 수를 비교하기 때문에 이진 비교기는 4개의 input을 가지고 a>b ,a=b, a<b 를 조사하기 때문에 3개의 output을 가진다.그에 따른 진리표를 작성하면 다음과 같다. 세 개의 output은 각각 f1, f2,f3 라 이름붙였다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| a1 | a2 | b1 | B2 | a>b | a=b | a<b |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

위 진리표에 따라 카르노 맵을 작성하면 다음과 같다.

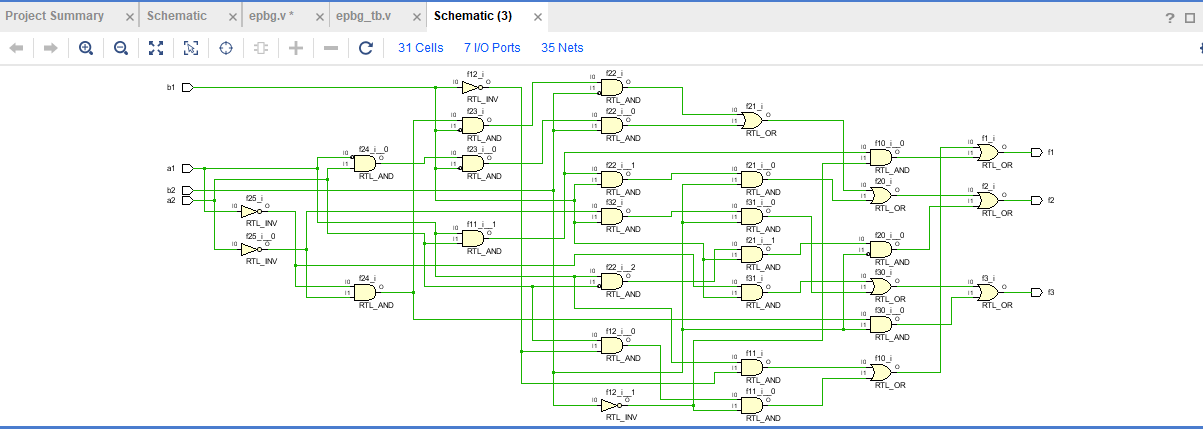


위 카르노 맵에서 얻은 식을 토대로 Verilog 식을 세우면 다음과 같다.

텍스트, 편지이(가) 표시된 사진

자동 생성된 설명

위 코드의 elaborated design을 확인하면 다음과 같다.



또한 simulation 결과는 다음과 같다.

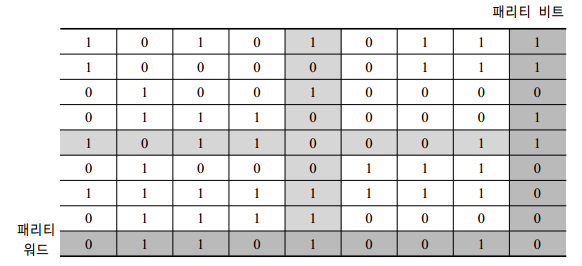


의도한대로 a가 00 b가 10일 경우 a보다 b가 크므로 a>b인 f1의 값이 0이고 a=b인 f2의 값 역시 성립하지 않기 때문에 0으로 나타나고 a<b의 값인 f3가 성립하기 때문에 1의 값을 가지는 것을 알 수 있다.

**4.** even barity bit 생성기와 even barity bit 점검기, odd barity bit 생성기와 odd barity bit 점검기의 진리표를 작성하고 이에 대한 카르노 맵을 작성해서 식을 정리했다. 정리된 식을 Verilog 코드를 통해 코드를 구현하고 이를 testbench 소스를 통해 직접 simulation 결과를 눈으로 확인할 수 있었다. 또한 even pritiy bit 점검기와 생성기와 odd parity bit 생성기와 점검기가 서로 반대된 결과를 가지는 것을 시뮬레이션 결과를 통해 확인할 수 있었다.

2 bit 이진 비교기 역시 진리표를 작성하고 이를 토대로 카르노 맵을 만들어 식을 정리했다. 이 식을 Verilog를 통해 구현하고 simulation 결과를 확인할 수 있었다. 또한 시뮬레이션 결과를 보며 설정한 변수 f1, f2, f3 가 a>b,a=b,a<b는 두 조건이 만족할 수 없기 때문에 서로 겹치는 경우가 단 하나도 없는 것을 확인할 수가 있다.

**5.** parity bit 의 단점으로는 오류를 점검해 오류가 있음을 확인은 할 수가 있지만 그 오류에 대해 수정하는 것은 불가능하다는 단점이 있다. 이러한 parity bit의 단점을 보완하기 위해 보완된 방식이 병렬 parity bit다. 병렬 parity 비트는 parity bit를 가로 세로로 구성되는 블록으로 구성해 오류를 점검하고 수정할 수 있게 한다.



오류가 발생하면 위와 같이 두 parity bit가 겹치는 부분에서 오류가 발생했다는 것을 알 수 있기 때문에 위치를 알 수 있다. 오류가 나타난 bit의 정확한 위치를 알 수 있기 때문에 이를 수정하는 것이 가능해진다.