**7주차 결과보고서**

**전공: 경영학과 학년: 3학년 학번: 20190963 이름: 한다현**

**1.**

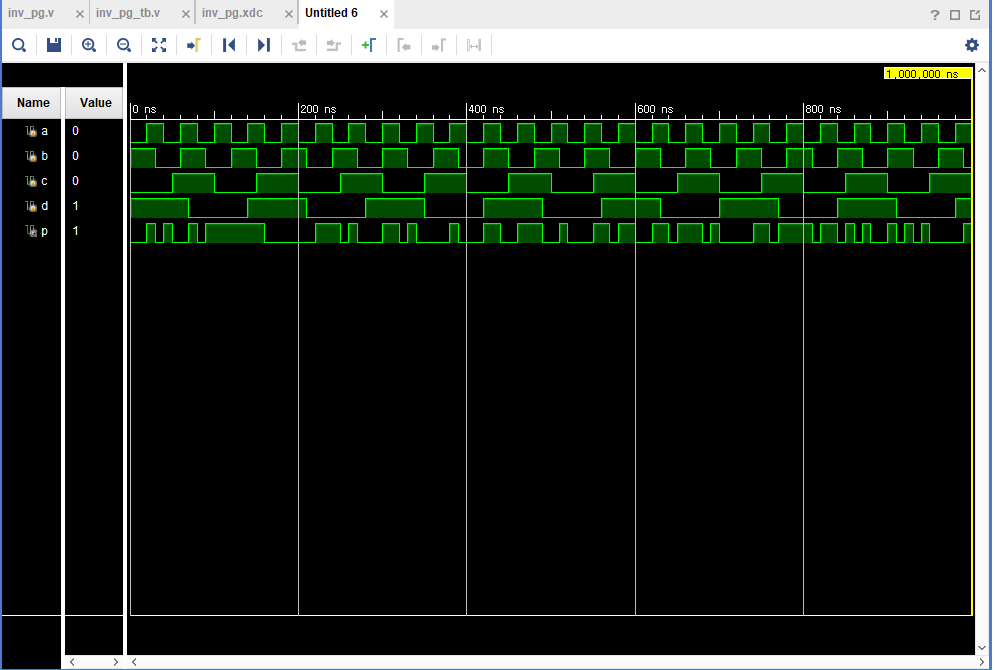
**Even parity bit generator는 데이터에 1의 개수가 홀수일 때는 parity bit을 1로 설정하여 데이터에 추가하고 데이터에 1의 개수가 짝수일 때는 parity bit을 0으로 설정하여 데이터에 추가하여 parity bit을 포함한 데이터에 1의 개수가 항상 짝수가 되도록 parity bit을 생성하는 기능을 한다. 따라서 a, b, c, d를 입력 받는 even parity bit generator를 구현하기 위해서 진리표와 카르노 맵을 작성한 후 식을 도출하여 정리하면 다음과 같다.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input a** | **Input b** | **Input c** | **Input d** | **Output p** |
| **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** | **1** | **0** |

테이블이(가) 표시된 사진

자동 생성된 설명

**입력 a, b, c, d가 일정한 간격으로 변화하게 설계한 후 시뮬레이션을 통해 변화하는 p의 값을 확인할 수 있다.**



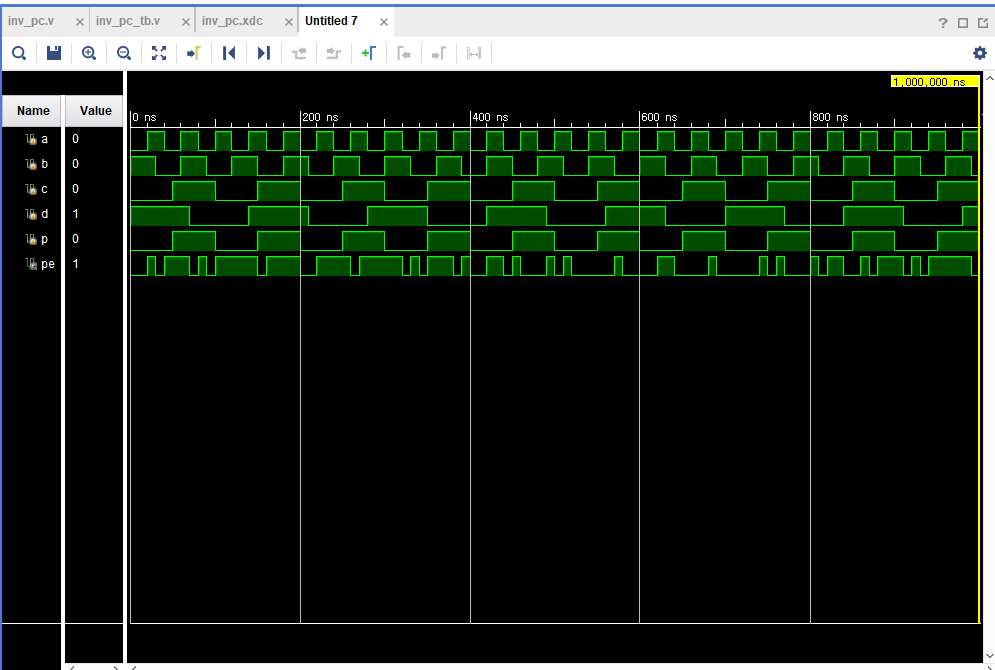
**Even parity bit generator를 통해 전달받은 데이터에 오류가 발생하였는지 확인하는 장치가 even parity bit checker이다. Even parity bit checker는 입력 a, b, c, d와 parity bit 중 1의 개수가 짝수이면 0을 pec(parity error checker)로 출력하고, 1의 개수가 홀수이면 1을 pec로 출력한다. 즉 even parity bit checker의 출력이 1이라면 오류가 발생한 것이다. 인풋 a, b, c, d, p를 입력 받는 Even parity bit checker를 구현하기 위해 진리표와 카르노 맵을 작성하고 식을 정리하면 다음과 같다.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input p** | **Input a** | **Input b** | **Input c** | **Input d** | **Output pec** |
| **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** |

테이블이(가) 표시된 사진

자동 생성된 설명

**Pec를 위 식과 같이 설계하고 a, b, c, d, p가 일정한 간격으로 변화하도록 설정한 후 시뮬레이션을 동작시키면 다음과 같이 pec가 변화하는 결과를 볼 수 있다.**



**2.**

**Odd parity bit generator는 데이터에 있는 1의 개수가 짝수이면 parity bit 1을 추가하고, 1의 개수가 홀수이며 parity bit 0을 추가하여 데이터에 있는 1의 개수가 항상 1이 되도록 parity bit을 생성하는 기능을 한다. 따라서 추가되는 parity bit의 값이 항상 even parity bit generator에 의해 생성된 parity bit의 역수이고 그에 따라 진리표와 카르노 맵 모두 even parity bit generator의 역이다. Odd parity bit generator의 진리표와 카르노 맵은 다음과 같다.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input a** | **Input b** | **Input c** | **Input d** | **Output 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** | **1** | **1** |

텍스트이(가) 표시된 사진

자동 생성된 설명

**입력 a, b, c, d가 일정한 간격으로 변화하도록 설계한 후 시뮬레이션 결과를 확인하면 다음과 같다. a, b, c, d 중 1의 개수가 홀수일 때 p는 0, 짝수일 때 p는 1이 출력되는 것을 확인할 수 있다.**

텍스트, 스크린샷, 전자기기, 디스플레이이(가) 표시된 사진

자동 생성된 설명

**Odd parity bit checker는 odd parity bit generator에게 전달받은 데이터의 오류를 검사하는 기능을 하는데, parity bit을 포함한 데이터에서의 1의 개수가 홀수이면 pec 0을 출력하고 1의 개수가 짝수이면 pec 1을 출력한다. a, b, c, d, p를 입력 받고 pec를 출력하는 odd parity bit checker의 진리표와 카르노 맵은 다음과 같다.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input p** | **Input a** | **Input b** | **Input c** | **Input d** | **Output pec** |
| **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** |

테이블이(가) 표시된 사진

자동 생성된 설명

**입력 a, b, c, d, p가 일정한 간격으로 변화하도록 설계한 뒤 시뮬레이션을 통해 pec의 결과를 확인하면 다음과 같다. Pec가 입력 중 1의 개수가 홀수이면 0, 짝수이면 1이 되는 결과를 확인할 수 있다.**

텍스트, 스크린샷, 전자기기, 디스플레이이(가) 표시된 사진

자동 생성된 설명

**3.**

**2 bit binary comparator는 2 bit 이진수 A, B를 비교한 결과를 출력한다. 출력 e, f, g가 있을 때, A가 B보다 크면 e를 출력하고 두 수가 같으면 f를 출력하고 B가 A보다 크면 g를 출력한다. A의 2 bit를 각각 a, b로 표현하고 B의 2 bit를 각각 c, d로 표현하여 진리표와 카르노 맵을 작성하면 다음과 같다.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **In a** | **In b** | **In c** | **In d** | **Out e** | **Out f** | **Out g** |
| **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** |

텍스트, 전자기기이(가) 표시된 사진

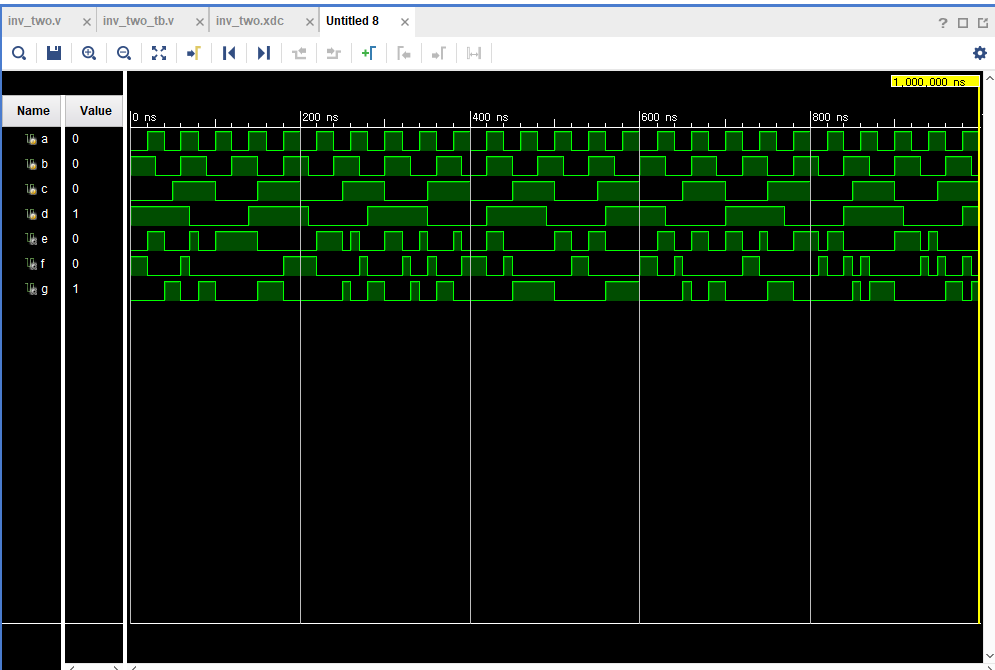
자동 생성된 설명

**위의 식들을 verilog에서 설계하면 다음과 같은 과정을 거쳐 출력이 결정되며**

텍스트, 스크린샷, 실내이(가) 표시된 사진

자동 생성된 설명

**입력 a, b, c, d를 일정한 간격마다 변화하도록 설정한 후 시뮬레이션 결과를 확인하면 다음과 같다.**



**4.**

**Even parity bit generator는 네 개의 인풋 a, b, c, d와 하나의 아웃풋 p를 사용하여 실험을 진행했다. 진리표와 카르노 맵을 이용해 p의 출력을 디자인하였고 시뮬레이션 결과 a, b, c, d 중 1의 값을 갖는 인풋의 개수가 짝수이면 p는 0이 출력되었고 홀수이면 p는 1이 출력되었다. Even parity bit checker는 기존 인풋 a, b, c, d와 추가된 parity bit인 p를 입력 받아서 오류 여부를 의미하는 pec를 출력하였다. 시뮬레이션 결과를 통해 a, b, c, d, p 중 1의 값을 갖는 인풋의 숫자가 짝수이면 pec는 0이 출력되었고, 홀수이면 1이 출력되었다.**

**Odd parity bit generator도 마찬가지로 네 개의 인풋 a, b, c, d와 하나의 아웃풋 p를 사용하여 실험을 진행하였다. 진리표와 시뮬레이션 결과를 확인해 봤을 때, odd parity bit generator의 parity bit은 even parity bit generator의 parity bit과 반대의 값을 취하는 것을 확인할 수 있었다. 즉 a, b, c, d 중 1의 값을 갖는 인풋의 개수가 짝수이면 p는 1이 출력되고, 홀수이면 0이 출력되었다. Odd parity bit checker도 인풋 a, b, c, d, p와 아웃풋 pec를 사용하여 실험을 진행하였는데, generator와 마찬가지로 even parity bit checker의 pec와 반대되는 pec 값을 출력하였다. 즉 a, b, c, d, p 중 1의 값을 갖는 인풋의 개수가 짝수이면 pec는 1을 출력하고, 홀수이면 0을 출력하였다.**

**마지막으로 2 bit comparator의 경우 A, B 두 2 bit 이진수를 사용하여 실험을 진행하였는데, A를 나타내는 인풋 a, b와 B를 나타내는 인풋 c, d를 입력하여 결과를 출력하였다. ab가 cd보다 클 경우에는 e를, 같을 경우에는 f를, 작을 경우에는 g를 출력하도록 디자인하였고, 시뮬레이션 결과 올바르게 출력되는 것을 확인할 수 있었다.**

**5.**

**기존의 parity bit checker로 데이터의 오류를 검사하는 방식과 다르게 수정된 parity bit 기술이 있다. 수정된 방법은 데이터를 전달하는 송신자가 데이터에 1개의 비트를 추가하는 것부터 시작한다. 이렇게 추가된 패리티 비트는 0 또는 1의 값을 가지며, 기존 데이터와 함께 수신기에 전달된다. 수신기는 송신기와 동일한 방법으로 전달받은 데이터를 디코딩한다. 디코딩을 통해 얻은 결과를 전송된 데이터와 비교하여 그 결과가 서로 같다면 오류가 없는 것으로 판단하고 같지 않다면 데이터에 오류가 발생했다고 판단한다.**

**참조:**

Modified Bit Parity Technique for Error Detection of 8 Bit Data