7주차 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20231523 이름: 김민정

**1. even parity bit generator and checker (+simulation, truth table, k-map)**

even parity bit generator는 error detection을 위해 사용하는 parity bit를 이용해 data bit와 parity bit를 합쳐 총 1의 개수가 짝수가 될 수 있게 만드는 시스템이다. 이에 관한 논리식은 truth table -> K-map -> 논리식 순서로 구할 수 있다. 먼저 진리표를 그려보자. 아래는 even parity bit generator의 truth table이다

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Truth-table | | | | |
| A | B | C | D | 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 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

아래는 위 truth table을 통해 그린 K-map과 논리식이다.

텍스트, 도표, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

마지막으로 구한 논리식을 통해 even parity bit generator가 어떻게 작동되는지 simulation으로 확인해보자. 아래는 even parity bit generator의 simulation이다. 여기서 aa, bb, cc, dd는 위 논리식에서 입력인 a, b, c, d를 의미하고 pp는 출력인 p를 의미한다.

|  |
| --- |
| simulation |
| 스크린샷이(가) 표시된 사진  자동 생성된 설명 |

이제 even parity bit checker에 대해 서술해보겠다. even parity bit checker는 송신한 data bit에 error가 있는지 detection할 수 있게 수신기에 포함되어있는 기기이다. input으로 a, b, c, d, p를 받고 이후 error가 존재하는지를 pec라는 출력을 통해 확인한다. Generator를 제작할 때와 동일한 과정으로 논리식을 구해보자. 아래는 even parity bit check truth table이다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Truth table | | | | | |
| A | B | C | D | P | 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 |

아래는 위 truth table을 통해 그린 K-map과 이를 통해 얻은 논리식이다.

텍스트, 도표, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

마지막으로 구한 논리식을 통해 even parity bit checker가 어떻게 작동되는지 simulation으로 확인해보자. 아래는 even parity bit checker의 simulation이다. 여기서 aa, bb, cc, dd, pp는 위 논리식에서 입력인 a, b, c, d, p를 의미하고 ppec는 출력인 pec를 의미한다.

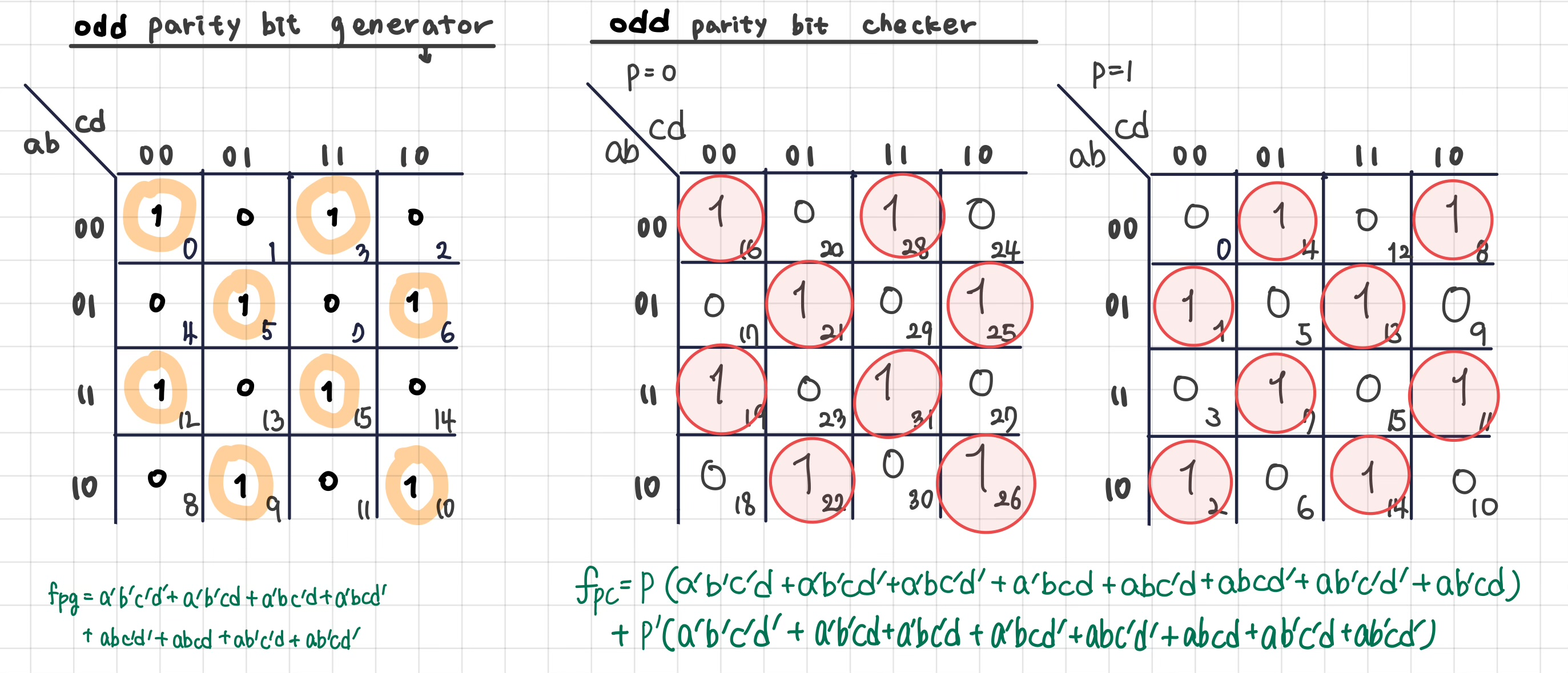
|  |
| --- |
| simulation |
| 스크린샷이(가) 표시된 사진  자동 생성된 설명 |

**2. odd parity bit generator and checker (+simulation, truth table, k-map)**

odd parity bit generator는 error detection을 위해 사용하는 parity bit를 이용해 data bit와 parity bit를 합쳐 총 1의 개수가 홀수가 될 수 있게 만드는 시스템이다. 이에 관한 논리식은 truth table -> K-map -> 논리식 순서로 구할 수 있다. 먼저 진리표를 그려보자. 아래는 odd parity bit generator의 truth table이다

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Truth-table | | | | |
| 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 |

아래는 위 truth table을 통해 그린 K-map과 논리식이다.



마지막으로 구한 논리식을 통해 odd parity bit generator가 어떻게 작동되는지 simulation으로 확인해보자. 아래는 odd parity bit generator의 simulation이다. 여기서 aa, bb, cc, dd는 위 논리식에서 입력인 a, b, c, d를 의미하고 pp는 출력인 p를 의미한다.

|  |
| --- |
| simulation |
|  |

이제 odd parity bit checker에 대해 서술해보겠다. odd parity bit checker는 송신한 data bit에 error가 있는지 detection할 수 있게 수신기에 포함 되어있는 기기이다. input으로 a, b, c, d, p를 받고 이후 error가 존재하는지를 pec라는 출력을 통해 확인한다. Generator를 제작할 때와 동일한 과정으로 논리식을 구해보자. 아래는 odd parity bit check truth table이다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Truth table | | | | | |
| A | B | C | D | P | 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 |

아래는 위 truth table을 통해 그린 K-map과 이를 통해 얻은 논리식이다.

텍스트, 도표, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

마지막으로 구한 논리식을 통해 odd parity bit checker가 어떻게 작동되는지 simulation으로 확인해보자. 아래는 odd parity bit checker의 simulation이다. 여기서 aa, bb, cc, dd, pp는 위 논리식에서 입력인 a, b, c, d, p를 의미하고 ppec는 출력인 pec를 의미한다.

|  |
| --- |
| simulation |
|  |

**3. 2-bit binary comparator (+simulation, truth table, k-map)**

2-bit binary comparator는 2-bit로 들어온 input 두개를 비교하여 왼쪽 요소를 기준으로 equal, big, small(오른쪽을 기준으로 삼아도 상관없긴 하다)을 판별하는 비교기이다. 이에 관한 논리식은 truth table -> K-map -> 논리식 순서로 구할 수 있다. 먼저 진리표를 그려보자. 아래는 2-bit binary comparator의 truth table이다

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Truth table | | | | | | |
| a1 | a2 | b1 | b2 | Equal | Big | Small |
| 0 | 0 | 0 | 0 | 1 | 0 | 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 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |

아래는 위 truth table을 통해 그린 K-map과 이를 통해 얻은 논리식이다. 이를 통해 equal은 a1’a2’b1’b2 + a1’a2b1’b2 + a1a2b1b2 + a1a2’b1b2’, big은 a1b1’ + a2b1’b2’ + a1a2b2’, 마지막으로 small은 a1’a2’b2 + a1’b1 + a2’b1b2를 통해 구할 수 있다.

텍스트, 스크린샷, 번호, 도표이(가) 표시된 사진

자동 생성된 설명

마지막으로 구한 논리식을 통해 2-bit binary comparator가 어떻게 작동되는지 simulation으로 확인해보자. 아래는 2-bit binary comparator의 simulation이다. 여기서 aa1, aa2, bb1, bb2는 위 논리식에서 입력인 a1, a2, b1, b2를 의미하고 llt, ggt, eeq는 출력인 small, big, equal를 의미한다.

|  |
| --- |
| simulation |
| 스크린샷, 사각형, 멀티미디어 소프트웨어이(가) 표시된 사진  자동 생성된 설명 |

**4. 결과 검토 및 논의 사항**

Parity bit generator와 checker 그리고 2-bit comparator를 구현하며 진리표와 K-map을 이용한다면 대부분의 논리식을 구할 수 있음을 확인할 수 있었다. 하지만 K-map 변수가 5개 이상이 된다면 차원이 많아져 고려해야 할 implicant가 복잡해지기에 6개 이상의 변수에는 k-map 대신에 다른 방도를 사용해야 할 것 같다.

**5. 추가 이론 조사(Hamming code)**

Parity bit는 error detection밖에 구현할 수밖에 없어, 오류가 발생했음만을 확인할 수 있고 오류가 발생한 부분을 찾을 수 없었다. 이와 달리 Hamming code는 error detection뿐만아니라 error correction을 할 수 있어, 틀린 부분이 발생하면 오류와 더불어 어디서 오류가 발생했는지 확인할 수 있다. 예시를 들어보자. 만약 4개의 data bit를 3개의 check bit를 이용해서 error correction을 진행해보자. 만약 수신기에서 받은 code를 0010011이면 check bit e1=0, e2=1, e4=0이다.(e1, e2, e4는 자기 자신과 hamming code가 같은 지 다른 지 확인하는 논리식이다.) 이러면 bit error는 1\*0 + 2\*1 + 4\*0 = 2번째 비트가 틀린 것이 된다. 그러니 원래의 송신기에서 보냈던 코드는 0110011이 되며, 원래의 데이터 비트는 1011이 된다.